Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tomaž Marinčič

Orodje za snovanje trojiških kvantnih celičnih avtomatov

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Iztok Lebar Bajec

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V diplomski nalogi preučite obstoječe orodje za simuliranje trojiških kvantnih celičnih avtomatov in zasnujte programsko opremo za vizualno snovanje takih sistemov. Osredotočite se na grafični uporabniški vmesnik, ki bo uporabniku omogočal čim lažje snovanje struktur, simulacijo in analizo.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Tomaž Marinčič, z vpisno številko 63110152 sem avtor diplomskega dela z naslovom:

Orodje za snovanje trojiških QCA struktur

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Iztoka Lebarja Bajca,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 30. januarja 2015 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Iztoku Lebarju Bajcu za vodenje pri izdelavi in razložitvi teme diplomske naloge ter za strokovne nasvete pri ustvarjanju orodja. Zahvaljujem se tudi svoji družini za mojo podporo skozi celoten študij.

That so few dare to be eccentric, marks the chief danger of our time.
- - John Stuart Mill

Kazalo

Povzetek

Tehnologija kvantnih celičnih avtomatov (angl. *Quantum-dot Cellular Automata - QCA*) ponuja alternativo osnovnim gradnikom tehnologije CMOS, tranzistorjem. Pri tem izkoristi kvantne učinke, katerim se tehnologija CMOS poskuša izogniti. Prvotno se je razvoj osredotočil na dvojiški procesni QCA model. Zadnja leta pa se je usmeril tudi v razvoj trojiškega procesnega QCA modela. Njegov osnovni gradnik je trojiška QCA celica, ki je splošna razširitev dvojiške QCA celice. Te se v modelu razporejajo v strukture, ki z medsebojnimi interakcijami omogočajo izvedbo procesiranja.

Cilj diplomske naloge je računalniško orodje, ki omogoča urejanje poljubne trojiške QCA strukture in s tem raziskovanje in razširjanje funkcijskega nabora trojiškega QCA modela.

Orodje smo razvili kot namizno aplikacijo, v kateri lahko sestavimo poljubno trojiško QCA strukturo, simuliramo njeno obnašanje in pregledamo rezultate simulacije. Aplikacija vpogled v svet trojiških QCA struktur nudi preko grafičnega vmesnika, ki lastnosti strukture predstavlja kot geometrične oblike različnih lastnosti. Aplikacija je bila narejena v programskem okolju Visual Studio, v jeziku C++. Za ogrodje arhitekture ter matematične in grafične podpore programa smo uporabili programsko knjižnico Cinder.

Ključne besede: kvantni celični avtomati, procesiranje, struktura, aplikacija.

Abstract

The technology of Quantum-dot Cellular Automata (QCA) offers an alternative to the basic constructs of CMOS, transistors. It takes advantage of quantum effects, which are avoided in CMOS technology. Originally the development focused on the binary QCA processing model. In the last few years the research focused on the ternary QCA model. Its basic construct is the ternary QCA cell, which is an expansion of the binary QCA cell. They are put in a structure, where their interactions enable logical processing.

The goal of this thesis is a computer tool that enables us to build any QCA structure, simulate its behaviour and then analyse the results. The application offers an insight into ternary QCA structures via a graphical interface, which presents raw data graphically as geometric shapes with different properties. The application was developed with C++ in Visual Studio. The Cinder library was used to form the application's basic skeleton and offered graphical and mathematical support.

Keywords: quantum-dot cellular automata, processing, structure, application.

Poglavje 1

Uvod

1.1 Motivacija

Trenutna vodilna tehnologija v računalniških čipih je CMOS (angl. *Complementary Metal Oxide Semiconductor*). Že od splavitve tehnologije smo opazovali njeno eksponentno rast v praktično vseh področjih zmogljivosti. Sedaj opažamo znake upočasnjevanja, ali celo ustavitve napredka. Razlog temu so dosežene tehnološke in teoretične omejitve. Ta ustavitev predstavlja priložnost za napredek alternativnih tehnologij [10].

Med glavnimi razlogi za tehnološko omejitev CMOS tehnologij sta problem proizvedene toplote in problem povezovanja osnovnih gradnikov. Razvoj CMOS tehnologije temelji na povečevanju gostote tranzistorjev v prostoru ter povečevanju hitrosti preklopa le teh. Obe spremembi povečata količino proizvedene toplote, katere ne moremo odvesti dovolj hitro. Povečanje temperature povzroči povečanje količine šuma, kar negativno vpliva na zmogljivost.

Problem povezovanja osnovnih gradnikov izhaja iz dejstva, da kovinskih povezav med tranzistorji ne moremo manjšati z enakim tempom kot velikost samih tranzistorjev. Teoretične meje postanejo očitne, ko osnovni gradniki preidejo na velikostni red nanometra. Ta namreč sovpada z velikostnim redom molekul in atomov. V tem območju začnejo kvantni učinki močno

vplivati na obnašanje osnovnega gradnika, česar pa dosedanja tehnologija ni obvladovala, oziroma se je temu poskušala izogniti [8, 10].

Možen način, da mikrotehnološka industrija ohranja napredek v gostoti naprav je, da preklopi iz FET-bazirane (angl. *Field-Effect Transistor*) paradigme na eno, zasnovano na nanostrukturah. Pri tej paradigmi se kvantnim učinkom ne poskušamo izogniti, ampak jih izkoristimo sebi v prid. Eno izmed nanostrukturnih paradigm je predlagal C. S. Lent. Ta se imenuje kvantni celični avtomati (angl. *Quantum-dot Cellular Automata - QCA*) [12].

V računalniškem svetu so znane mnoge prednosti večvrednostne logike. Ljudje štejejo v desetiškem sistemu zaradi kulturnih razlogov, kot je ta, da imamo deset prstov. V inženirskem svetu se je uveljavil dvojiški sistem zaradi preprostosti in zanesljivosti dvojiškega računalniškega vezja. V svojem delu, Hayes [3] predstavi prednosti trojiškega številčnega sistema. Ta za razliko od dvojiškega in desetiškega sistema vsebuje zanimive matematične lastnosti. Med drugim je najbolj ekonomičen način predstavitve števil od vseh številskih sistemov s celoštevilskimi bazami. Te lastnosti se lahko izrabijo za pohitritev osnovnih operacij kompleksnejših algoritmov in struktur, kot je hitrejše preiskovanje iskalnega drevesa.

Osnovni gradnik današnjih računalniških sistemov je tranzistor, ki je dvostopenjsko tokovno stikalo. V začetnih časih razvoja računalniških sistemov so se pokazale tehnološke omejitve večvrednostnih osnovnih gradnikov, kar je privedlo do prevlade dvojiških sistemov.

Razvoju se je prilagodil tudi Lent, ki je osnovni gradnik računalniških sistemov razvil kot nadomestek tranzistorjem. To je QCA celica, ki lahko hrani dva energetsko ekvivalentna stanja in s tem hrani eno dvojiško vrednost. Celice, razporejene v strukturo tvorijo QCA strukturo, ki omogoča izvedbo procesiranja oziroma želeno spremembo stanja.

Lebar Bajec s sodelavci [4] je predstavil razširitev dvojiške celice s štirimi kvantnimi pikami v razširjeno ali trojiško QCA celico (angl. *ternary QCA - tQCA*) z osmimi kvantnimi pikami. Ta omogoča razširitev možnih logičnih stanj iz dveh na tri, kar omogoča predstavljanje trojiške logike v eni QCA

celici [4]. V tem delu te celice imenujemo trojiške QCA celice.

V pričujočem delu je opisano programsko orodje, ki omogoča sestavljanje, urejanje, simuliranje in pregled rezultatov simulacije poljubnih trojiških QCA struktur. Za uporabo programa je potrebno osnovno znanje o QCA, ki je v delu tudi opisano.

1.2 Metodologija

Kot opisano zgoraj je cilj naloge izdelava orodja za snovanje trojiških QCA struktur. Na začetku razvoja je bila potrebna pridobitev znanja o kvantnih celičnih avtomatih. To znanje smo, po nasvetu mentorja, pridobili primarno iz magistrskega dela Primoža Pečarja, [8], ki dobro opiše parametre celic, ter način procesiranja.

Program je razvit v razvojnem okolju Visual Studio 2012 ter programskem ogrodju Cinder, opisan v poglavju 4.2. Za uporabniški vmesnik je tudi pripravljena knjižnjica za uporabniški vmesnik ciUI (angl. *Cinder User Interface*), opisan v poglavju 4.3. Vse tri stvari smo spoznali ob času uporabe. Torej Visual Studio 2012 in Cinder pred začetkom razvoja, uporabniški vmesnik ciUI pa ob začetku razvoja uporabniškega vmesnika.

Ključen del orodja je sam simulator, ki je bil razvit v okviru LRSS. Vhodne in izhodne datoteke tega simulatorja so zadoščale za razvoj arhitekture vnaprej, prav tako so nudile vpogled v želeno ciljno funkcionalnost programa. Ta se direktno nanaša na spreminjanje želenih parametrov vhodne datoteke ter predstavitev rezultatov izhodne datoteke. Te datoteke smo spoznali pred samim razvojem.

Po končanem programiranju smo se osredotočili na pričujoče delo, v katerem smo predstavili zbrano znanje ter opisali izdelano orodje.

1.3 Pregled dela

To delo služi kot dokumentacija orodja za urejanje trojiških QCA struktur. Predstavljeno je tudi znanje o kvantnih celičnih avtomatih, ki je bilo potrebno za razvoj orodja.

V drugem poglavju predstavimo kvantne celične avtomate kot celice s kvantnimi pikami. To znanje razširimo z uporabljenim modelom procesiranja kvantnih celičnih avtomatov. V četrtem poglavju opišemo delovanje in vsebovano funkcionalnost razvitega orodja. Nato v petem poglavju predstavimo arhitekturo in programsko okovje. Šesto poglavje vsebuje opis uporabljenih orodij pri razvoju.

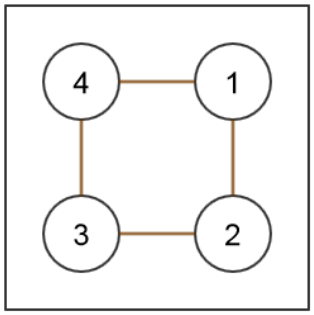
Poglavje 2

Kvantni celični avtomat

QCA ali kvantni celični avtomat je naprava, ki jo je v svojem članku [6] predstavil C. S. Lent s sodelavci. Podreja se zakonom kvantne mehanike [1] in teoriji celičnih avtomatov [13]. Naprava je podana kot polje celic, ki omogočajo predstavitev končnega števila stanj, medtem ko je stanje neke celice odvisno od stanja njenih sosed [2, 6]. Osnovni gradnik naprave je celica s kvantnimi pikami ali QCA celica.

2.1 Celica s kvantnimi pikami

Celica s kvantnimi pikami je naprava nanometrske velikosti, zgrajena iz določenega števila kvantnih pik, ki vsebuje kvantne delce. Kvantna pika je tridimenzionalna nanometrska struktura oz. področje, v katerem lahko lokaliziramo naboj. To je izvedeno z zajetjem (angl. *confinement*) kvantnega delca, v našem primeru elektrona, s pomočjo potencialnih pregrad (angl. *potential bariers*). Zaradi kvantno-mehanskih lastnosti zajeti elektron tunelira med sosednjimi kvantnimi pikami. Torej si QCA celico lahko predstavljamo kot planarno strukturo, v kateri se elektron prosto giblje le v območju kvantnih pik med katerimi lahko tunelira, izven tega področja pa ne more zaiti, prikazano na sliki 2.1. Celica deluje v režimu, kjer Coulombovi vplivi prevladujejo nad tuneliranjem, kar pomeni, da zajeti elektron, v primeru prisotnosti



Slika 2.1: Kvantna celica s štirimi pikami.

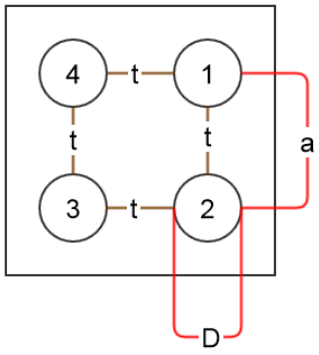
drugih nabojev v okolici čuti Coulombove sile in zaradi tega teži k lokalizaciji v eno izmed kvantnih pik celice [8].

Coulombov zakon trdi, da je absolutna vrednost sile premo sorazmerna produktu obeh nabojev in obratno sorazmerna s kvadratom razdalje med njima. V svetu kvantnih celičnih avtomatov si zakon lahko interpretiramo tako, da si predstavljamo, da elektroni silijo v pike, v katerih bodo najbolj oddaljeni od drugih elektronov. V takem stanju na njih vpliva najmanjša možna sila, kar pomeni, da imajo najmanjše možno energijsko stanje.

Celica je lahko sestavljena iz poljubnega števila pik in v njej je v splošnem lahko zajeto poljubno število elektronov.[1] Lent je raziskave, ki jih je opravljal v začetku 90-ih let prejšnjega stoletja, omejil na celico, ki bi bila sposobna predstavitve dvojiške informacije. Ta je sestavljena iz štirih kvantih pik nameščenih v vogale kvadrata. Gre za okrogle kvantne pike s premerom $D = 10\text{nm}$, ki so med seboj oddaljena $a = 20\text{nm}$, slika 2.2. V celici sta zajeta dva elektrona, ki lahko tunelirata le med sosednjimi pikami. V smislu celičnega avtomata razporeditev elektronov v kvantnih pikah določa stanje celice. Tuneliranje elektronov izven celice ni mogoče, saj se predpostavlja, da je popolnoma zadušeno s potencialnimi pregradami [8].

V izolirani QCA celici, torej celici, na katero ni prisotnih zunanjih vplivov,

[1] Seveda obstajajo tehnološke omejitve, a za razumevanje te niso pomembne.



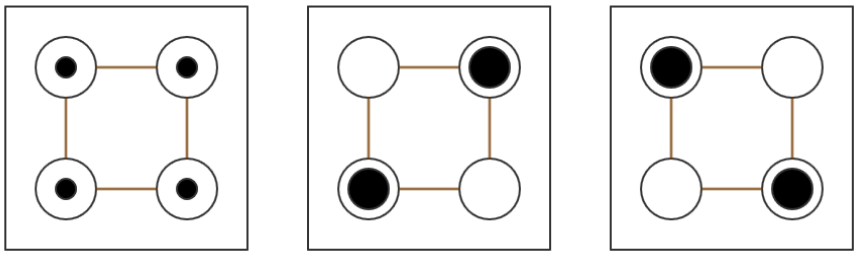
Slika 2.2: Kvantna celica s štirimi pikami, velikostjo pik $D = 10\text{nm}$ in razdaljo med središči pik $a = 20\text{nm}$.

se zajeta elektrona zaradi Coulombove odbojne sile skušata lokalizirati v pikah, ki sta medsebojno najbolj oddaljeni. V teh pikah je odbojna sila med elektronoma najmanjša. Torej se v primeru kvadratne razporeditve pik elektrona postavita diagonalno v vogale celice in v vsakem trenutku obstajata dve energijsko ekvivalentni razporeditvi, slika 2.3. Z drugimi besedami to pomeni, da je lokalizacija elektronov enako verjetna v vseh kvantnih pikah. Pravimo, da je celica tedaj v nevtralnem stanju.

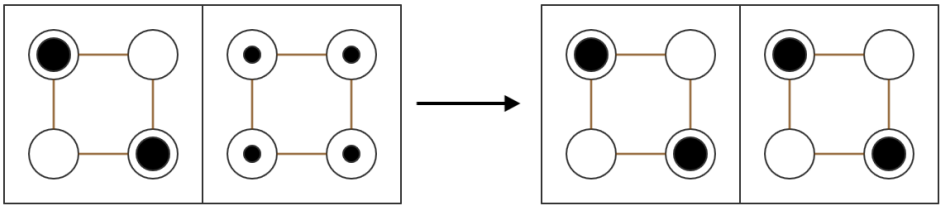
2.2 Interakcije med celicami

Interakcije med celicami še zmeraj potekajo samo preko odbojnih Coulombovih sil. Bližina druge celice povzroči energijsko ne-nevtralno stanje, ki se razreši z ustreznim tuneliranjem elektronov v stanje, ki se prilega drugi celici, kot prikazano na sliki 2.4. Pri tem primeru je tunelirna energija druge celice padla iz visoke vrednosti na nizko. Tako se je omogočilo tuneliranje med pikami, kar je privedlo do spremembe stanja, ki je sedaj energijsko stabilno. Na sliki 2.5 je primer, kjer sta celici navpično zamaknjeni. V tem primeru se izvede negacija stanja druge celice glede na prvo.

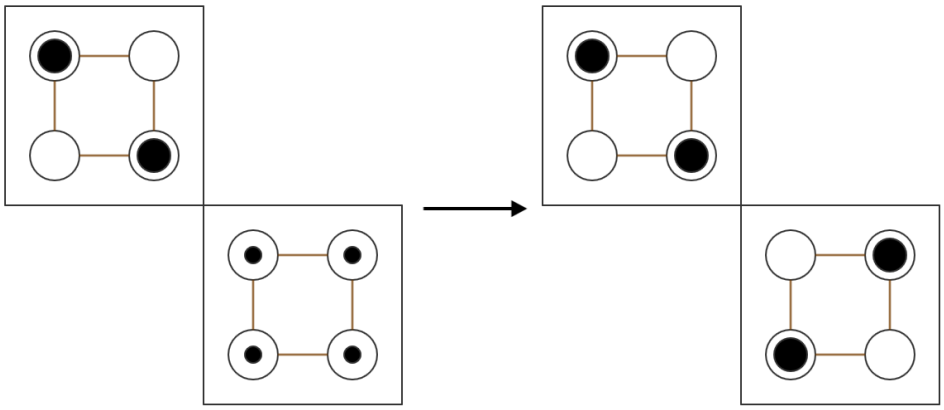
Torej vsa interakcija med celicami poteka le na osnovi Coulombovih sil,



Slika 2.3: Tri QCA celice. Prva je v energijsko nevtralnem stanju in prikazuje trenutek, ko sta možni obe diagonali. Drugi dve sta v dveh različnih, energijsko ekvivalentnih, stabilnih stanjih.



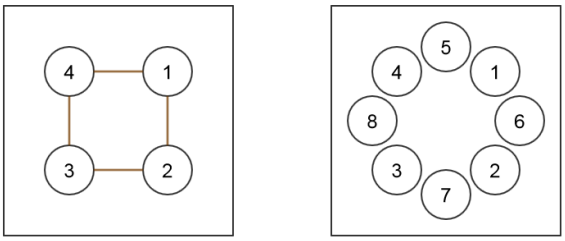
Slika 2.4: Sprememba stanja druge celice, ki po spremembi ustreza razporeditvi prve celice. Druga celica je pred prehodom v nevtralnem stanju.



Slika 2.5: Prehod stanja druge celice, v primeru, da so celice navpično zamaknjene.

pod vplivom katerih se zajeti elektroni razporejajo v celici. Ker tuneliranje elektronov med celicami ni možno, ni prisotnega električnega toka, kar močno vpliva na zmanjšanje porabe energije in njene disipacije v okolje v obliki toplote. To je ena izmed poglavitnih prednosti QCA arhitekture. Tako bi v primeru nadomestitve tranzistorja, kot osnovnega gradnika, s QCA celico rešili enega največjih problemov današnje VLSI CMOS tehnologije [8].

Zgoraj opisana interakcija med celicami je možna le, če se v vsaki celici zagotovi simetrična nevtralizacija naboja (angl. *Symmetric Charge Neutralization - SCN*). S tem se doseže, da je s stališča okolice celoten naboj enak nič, torej je celica navzven elektrostatično nevtralna in ne povzroča monopolnega električnega polja. Ob upoštevanju SCN vsaka točka v opazovani celici občuti isto absolutno elektrostatično silo druge celice, ne glede na razporeditev elektronov v njej. Torej, če neka točka v opazovani celici pri eni diagonalni razporeditvi nabojev čuti privlačno silo, bo pri nasprotni diagonalni razporeditvi občutila enako odbojno silo. To pomeni, da je odbojna sila, ki jo izkazuje neka celica enaka njeni privlačni sili [8].



Slika 2.6: Na levi dvojiška in na desni trojiška QCA celica, vsaka z označenimi številkami pik. Trojiška celica se šteje kot razširitev dvojiške, kjer med vsaki dve piki dvojiške celice dodamo novo piko.

2.3 Parametri celic

Celica je lahko sestavljena iz poljubnega števila pik in v njej je lahko zajeto poljubno število elektronov. V našem delu smo se omejili na simuliranje dvojiških in trojiških celic. Trojiška celica se šteje kot posplošitev dvojiške celice.

Po Lentu [6], je dvojiška celica sestavljena iz štirih kvantnih pik, razporejenih v vogale kvadratne celice. Kvantne pike imajo premer $D = 10\text{nm}$, razdalja med njihovimi središči je $a = 20\text{nm}$. Trojiška QCA celica je razširitev dvojiške QCA celice, kot prikazano na sliki 2.6.

Razdaljo med celicami, ki omogoča pravilen prenos stanja je Lent določil eksperimentalno, $r = 3a = 60\text{nm}$. Ta razdalja se ohranja pri trojiški celici.

Oznake kvantnih pik si sledijo po številki naraščajoče. Štetje se začne pri 1, v zgornji desni piki. V dvojiški celici se štetje nadaljuje v smeri urinega kazalca. Trojiška celica ohrani zaporedne številke dvojiške, štetje pik se nadaljuje z piko 5, ki je med pikami 1 in 4. Oznake so predstavljene na sliki 2.6.

2.3.1 Tunelirna energija - t

Parameter t se imenuje tunelirna energija, njen kvadrat je premo sorazmeren z verjetnostjo tuneliranja med sosednjimi pikami. Njegove vrednosti ležijo na intervalu [0,-2]. Tehnologija QCA omogoča spreminjanje tunelirne energije posamezne celice. Ta reši problem grobega preklapljanja, pri katerem lahko celice med prekljapljanjem v drugo stanje obtičijo v metastabilnem stanju, to je energijsko ne minimalno stanje [5]. Rešitev problema je ponudil Lent s sodelavci v [5], ki se imenuje adiabatno preklapljanje (angl. *adiabatic switching*). Pri tem zagotavljamo, da je struktura ves čas preklopa v trenutnem osnovnem stanju. Implementacija adiabatnega preklapljanja sestoji iz kontrolnega mehanizma v obliki urinega signala, ki izvaja sinhronizacijo prenosa podatkov med celicami. Urin signal ni neposredno priključen na celico, temveč vpliva nanjo v obliki električnega polja, ki določa višino potencialnih pregrad med kvantnimi pikami, ali tunelirne energije [8].

V našem orodju je spreminjanje višine potencialnih pregrad v celici omogočeno s spreminjanjem urine faze (angl. *clock phase*) posamezne celice. Ta določa začetno stanje urine periode. Stanja urine faze in črke njene oznake so sledeče [8]:

- Brezfazno stanje (angl. *Unclocked - U*). Celica ni vezana na urino periodo, njena tunelirna energija je konstantna in omogoča tuneliranje elektronov med pikami.

- Faza preklopa (angl. *Switch phase - S*) sledi fazi sproščenosti. Na začetku faze so potencialne pregrade spuščene, kar pomeni nizko stopnjo lokalizacije elektronov v kvantnih pikah in veliko verjetnost tuneliranja med njimi. Posledica tega je nevtralno stanje celice. V nadaljevanju se pričnejo pregrade počasi dvigovati in možnost tuneliranja se manjša. Elektroni se ustalijo v razporeditvi, ki zagotavlja najnižje energijsko stanje glede na okolico; tako v tej fazi pride do dejanske izvedbe procesiranja. Na koncu so potencialne pregrade tako visoke, da tuneliranje med kvantnimi pikami ni več mogoče in elektroni so lokali-

zirani v kvantnih pikah.

- Faza zadrževanja (angl. *hold phase* - H) sledi fazi preklopa. V tej fazi so pregrade ves čas popolnoma dvignjene, kar onemogoča tuneliranje; celice niso dovzetne na vplive ostalih celic in ne morejo spremeniti svojega stanja. Ko so izhodne celice v fazi zadrževanja, lahko preberemo njihova stanja.

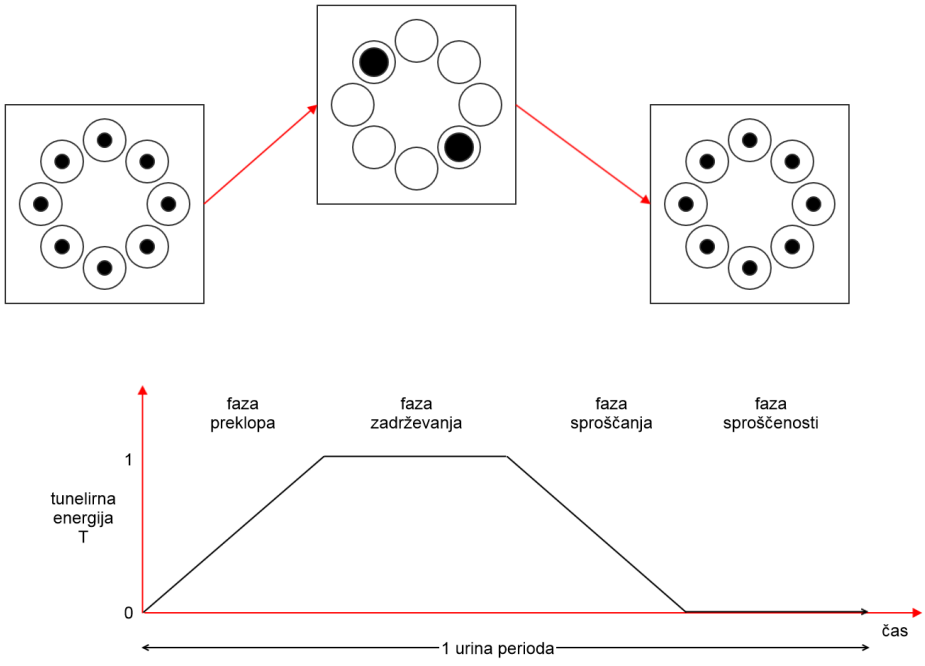
- Faza sproščanja (angl. *release phase* - R) sledi fazi zadrževanja. Potencialne pregrade se pričnejo spuščati in možnost tuneliranja se povečuje. Tako celica izgublja stanje, ki ga je predhodno zadržala in se sprošča v nevtralno stanje.

- Faza sproščenosti (angl. *relaxed phase* - L) sledi fazi sproščanja in je zadnja faza v ciklu. Potencialne pregrade so spuščene, kar omogoča veliko verjetnost tuneliranja elektronov med pikami, ki se kaže kot nevtralno stanje celice.

Urin signal, ki kontrolira te preklope je sestavljen iz zgoraj naštetih štirih faz. Višina pregrad je normalizirana na intervalu med 0 in 1. Pri tem 0 pomeni spuščene pregrade, v tem stanju velika verjetnost tuneliranja med pikami, 1 pa pomeni, da so pregrade dvignjene, kar onemogoča tuneliranje. Urine faze so enako dolge, tako vsaka zavzema četrtino urinega cikla, kot prikazano na sliki 2.7.

2.3.2 Gostota naboja v piki - ρ

Parameter ρ opisuje gostoto naboja v posamezni kvantni piki celice ali verjetnost, da se v neki piki nahaja elektron. Vrednosti segajo od 0, ki predstavlja odsotnost naboja ali nizko verjetnost prisotnosti elektrona, do 1, ki predstavlja polno količino naboja, ali popolno prisotnost elektrona.



Slika 2.7: Graf prehajanja urinih faz QCA celice, ki prikazuje vrednost tunelirne energije celice v odvisnosti od časa, opisano v poglavju 2.3.1. Sprehod čez vse faze traja eno urino periodo. Polarizacija celic se spreminja kot vidimo nad grafom, od visoke v fazi zadrževanja do nizke v fazi sprostitve.

2.3.3 Polarizacija P

Polarizacija P je skalarna količina, ki predstavlja verjetnost prisotnosti elektronov v vsaki posamezni diagonali celice. Vrednosti segajo od 0, ki predstavlja odsotnost elektronov, do 1, ki predstavlja prisotnost elektronov.

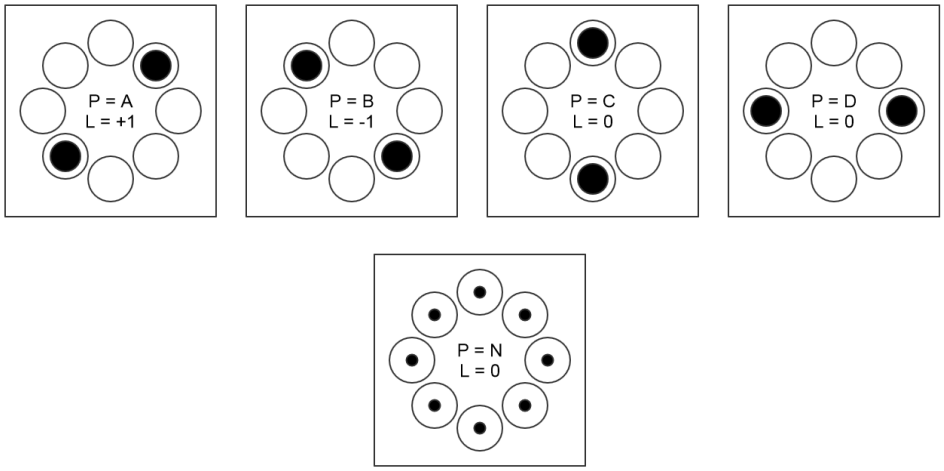
V našem orodju polarizacijo obravnavamo v dveh delih. Pri sestavljanju QCA strukture vsaki celici določimo začetno vrednost. Ta določa v katerih pikah se elektrona nahajata. Prisotnost elektronov v eni izmed diagonal označujemo z črkami $ABCD$, pri čemer A označuje prisotnost elektronov v pikah 1 in 3, B v pikah 2 in 4, C v pikah 5 in 7, D v pikah 6 in 8. Nevtralno stanje, v katerem sta elektrona enakomerno razporejena v vseh pikah celice, označimo z U. Na sliki 2.8 vidimo vse možne osnovne polarizacije.

Drugi del je pri predstavitvi rezultatov simulacije, kjer P predstavlja razporeditev elektronov v vsakem trenutku simulacije, kot opisano v prvem odstavku tega poglavja.

2.3.4 Logična vrednost - L

Kvantni celični avtomat kot procesna platforma potrebuje preslikavo fizičnih količin v logične vrednosti in obratno [9]. Logična vrednost je števka, ki enostavno opiše stanje osnovnega gradnika. Pri tehnologiji CMOS so to napetostna stanja, visoko stanje preslikamo v 0, nizko v 1. V prejšnjem poglavju 2.3.3 smo opisali polarizacijo celice. Kot opisano v članku [9], ta zadostuje za dvojiško celico, kjer polarizacijo neposredno preslikamo v logično vrednost.

Trojiške celice lahko zasedejo tri logične vrednosti, kot prikazano na sliki 2.8. Polarizacijo A preslikamo v $L = +1$ in B v $L = -1$. Obe polarizaciji, C in D, ter tudi nevtralno stanje N preslikamo v enako logično vrednost $L = 0$.



Slika 2.8: Možne polarizacije P in logične vrednosti L trojiške QCA celice.

2.4 Model procesiranja

Model procesiranja QCA je realiziran v obliki planarne strukture. Osnovni gradniki strukture so QCA celice. Glede na vlogo v strukturi ločimo štiri tipe celic [8]:

- vhodne celice - D[2] (angl. *Driver Cells*), so ponavadi postavljene na robovih strukture, in omogočajo prenos podatka v QCA.
- delovne celice - I (angl. *Internal cells*) se običajno nahajajo v strukturi in z medsebojno interakcijo realizirajo neko funkcijo oz podatkovno transformacijo.
- izhodne celice - O (angl. *Output cells*) so znova tipično postavljena na robovih strukture. Njihovo stanje se interpretira kot rezultat procesiranja.
- celice z vgrajenim stanjem - E (angl. *Electrode*) imajo stanje določeno že med procesom izdelave in se ne spreminja.

[2]D, I, O, E so v našem orodju oznake za tipe celic.

Na sliki 2.9 je primer trojiških majoritetnih vrat. Za to razporeditev celic velja pravilnostna tabela 2.1.

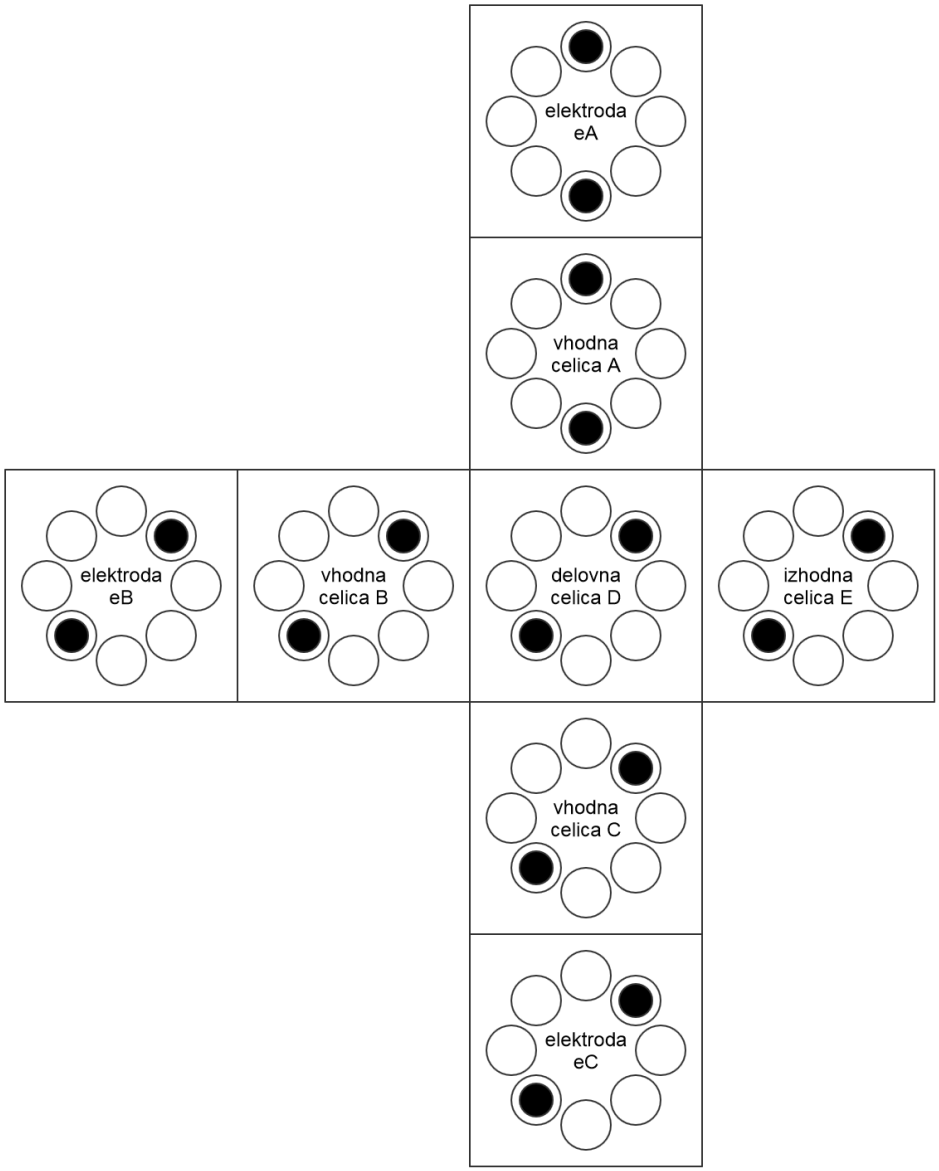
2.5 Simulator qdCAD

Orodje qdCAD - Quantum-dot Cellular Automata Designer, version 1.4.201-31111, (c) 2007-13, LRSS, FRI, UNI-LJ, je simulator medceličnih interakcij QCA struktur. Pri tem uporablja posplošeni Hubbardov QCA model, ki za izračun medceličnih interakcij uporablja postopek medcelične Hartree aproksimacije (angl. *InterCellular Hartree Approximation - ICHA).*

QCA celice so v Hubbardovem modelu predstavljene kot smo opisali v drugem poglavju 2. Torej kot celice s kvantnimi pikami, v katerih so zajeti elektroni, ki imajo možnost tuneliranja le med svojimi sosednjimi pikami. Interakcije elektronov v QCA strukturi so obravnavane s kvantnomehanskega stališča. Hubbardov model predstavlja alternativo realizacijsko enostavnejšem polklasičnem modelu, ki je, za razliko od Hubbardovega modela, nezmožen modeliranja adiabatnega pristopa, ki je opisan v poglavju 2.3.1.

V Hubbardovem modelu so medcelične interakcije obravnavane z izračunom ICHA. To je iterativni postopek, ki obravnava vsako posamezno QCA celico v odvisnosti od ostalih celic. Postopek v vsaki iteraciji izračuna novo stanje sistema, ki, kot smo opisali v poglavju 2.2, stremi k energijsko minimalnem stanju, kateremu se skozi iteracije približuje. Izračun novega stanja sistema je tako odvisen le od enega prejšnjega stanja sistema. Tega na začetku simulacije ustvarimo ročno, v našem primeru z našim orodjem, med izvajanjem simulacije pa se novo stanje računa glede na stanje, ki je rezultat prejšnje iteracije postopka.

Na začetku simulacije se stanje sistema opiše z določanjem polarizacije vsake celice in vseh ostalih lastnosti QCA strukture. Ta sistem nato v vsaj eni urini periodi, ki obsega nekaj iteracij ICHA postopka doseže neko stabilno, energijsko minimalno stanje. Nove spremembe se nato vnašajo le z



Slika 2.9: Primer trojiških majoritetnih vrat s tremi celicami z vgrajenim stanjem eA eB eC, tremi vhodnimi celicami ABC delovno celico D in izhodno celico E. Za to razporeditev celic velja pravilnostna tabela 2.1.

eA	eB	eC	E
-1	-1	-1	-1
-1	-1	0	-1
-1	-1	1	-1
-1	0	-1	-1
-1	0	0	0
-1	0	1	0
-1	1	-1	-1
-1	1	0	0
-1	1	1	1
0	-1	-1	-1
0	-1	0	0
0	-1	1	0
0	0	-1	0
0	0	0	0
0	0	1	0
0	1	-1	0
0	1	0	0
0	1	1	1
1	-1	-1	-1
1	-1	0	0
1	-1	1	1
1	0	-1	0
1	0	0	0
1	0	1	1
1	1	-1	1
1	1	0	1
1	1	1	1

Tabela 2.1: Pravilnostna tabela logičnih stanj majoritetnih vrat, prikazani na sliki 2.9. Če so vsaj dve vhodne vrednosti iste, se ta vrednost prenese na izhod. Če so vse vhodne vrednosti različne je izhod 0.

spreminjanjem polarizacij celic z vgrajenim stanjem, ki se nahajajo na robu strukture. Te spremembe si lahko predstavljamo kot vnos novih vhodnih podatkov za našo QCA strukturo kot procesni model. Imenujemo jih sekvenca vhodnih stanj.

Simulator qdCAD torej izvaja zgoraj opisani postopek spreminjanja stanja sistema. Pri tem stanje sistema v vsaki iteraciji shrani in vrne kot rezultat.

2.5.1 Vhodna datoteka

Vhodna datoteka qdCAD simulatorja s končnico `.qdStruct` hrani definicijo QCA strukture in parametre simulacije. Opis parametrov in polj vhodne datoteke, katere format je prikazan na izpisu 2.1, je sledeč:

Definicija QCA strukture:

- `nCells`
  Celoštevilska vrednost, ki določa število celic v QCA strukturi.

- `a x y z`
  Arhitektura `a` označuje velikost celice (60,72,110). Pozicije celic se množijo z množitelji medcelične razdalje `x y z`, ki so poljubne realne vrednosti.

- `type mode x y z phase value`
  Vsaka vrstica opisuje eno QCA celico s parametri, opisani v poglavju 2.3.
  `type` določa ali je celica dvojiška - B ali trojiška - T.
  `mode` določa tip celice: celica z vgrajenim stanjem - E, vhodna celica - D, delovna celica - I, izhodna celica O.
  `x y z` so realne vrednosti, ki določajo pozicijo celice.
  `phase` določa začetno urino fazo celice: brezfazno stanje - U, faza preklopa - S, faza zadrževanja - H, faza sproščanja - R, faza sprostitve - L.
  `value` določa začetno vrednost celice: $NABCD$.

Parametri simulacije in sekvenca vhodnih stanj:

- `nClocks`
  Celoštevilska vrednost, ki določa število urinih period simulacije.
- `ticsPerQuarterTock`
  Celoštevilska vrednosti, ki določa število iteracij ali število računanj novih stanj sistema na eno urino fazo.
- `eps`
  Realna vrednost, ki določa minimalno dovoljeno spremembo energije sistema znotraj ene iteracije simuliranja.
- `nSteps`
  Celoštevilska vrednost, ki določa maksimalno število korakov do dosege minimalne dovoljene spremembe energije sistema.
- `maxR`
  Realna vrednost, ki določa radij vpliva Coulombove sile.
- `nInputValues`
  Celoštevilska vrednost, ki določa število sekvenc vhodnih stanj.
- $nElectrodes * $ `inputValue`
  Sekvenca vhodnih stanj. Določa stanje vsake celice z vgrajenim stanjem vsako urino periodo. Zaloga vrednosti je $ABCD$.

Listing 2.1: Format vhodne datoteke simulatorja `tmpExport.qdStruct`.

```
1 nCells
2
3 a x y z
4
5 type mode x y z phase value (* nCells)
6
7
```

```
8  nClocks(long)
9  ticsPerQuarterTock(long)
10 eps(double)
11 nSteps(long)
12 maxR(double)
13 nInputValues(long)
14
15 $nElectrodes$ * inputValue(char(A/B/C/D])
```

2.5.2 Izhodna datoteka

To so datoteke s končnico `.qdSim`. Datoteka vsebuje rezultate simulacije. Vsaka vrstica opisuje stanje sistema ali QCA strukture v enem trenutku ali eni iteraciji računanja prehoda sistema v manjšo energijsko stanje. Format ene vrstice je:

`nCells *[nDots *[[dotX dotY dotZ] + [rho] + cellPhase + cellTunnelEnergy]] + nInputs + [inputs] + nOutputs + [outputs]`

Torej vsaka vrstica vsebuje pozicijo vsake pike vsake celice `dotX dotY dotZ`, njihove ρ vrednosti, trenutna faza `cellPhase`, ki lahko zavzame vrednosti $SHRL$ ali U, glej poglavje 2.3.1, tunelirna energija `cellTunnelEnergy` celice in številsko oznako vhodnih in izhodnih celic.

2.5.3 Izvajanje programa qdCAD

Simulator qdCAD je samostojni program, s katerim komuniciramo preko vhodne in izhodne datoteke. Vhodna datoteka je sestavljena iz dveh delov, definicije strukture in parametrov simulacije. Prvi del definira QCA strukturo, torej parametre vseh celic, opisane v poglavju 2.3. Parametri celic določijo začetno stanje sistema, ki smo ga opisali v prejšnjem poglavju. Drugi del pa vsebuje parametre simulacije, ki določajo potek celotne simulacije in sekvenco vhodnih stanj, ki smo jo opisali zgoraj.

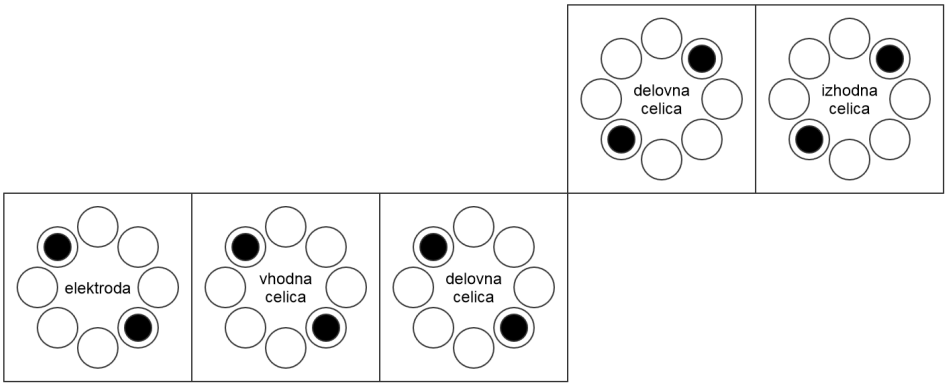
Program v ukazni vrstici zaženemo z ukazom `"qdCAD_sim.exe [input.`

`qdStruct] [output.qdSim]"`, ki ob koncu izvajanja pusti rezultate v datoteki z imenom `output.qdSim`. Format vhodne datoteke `input.qdStruct` in opis njenih parametrov je v poglavju 2.5.1.

Ročno simuliranje poljubne QCA strukture samo z programom qdCAD poteka tako, da najprej proizvedemo poljubno QCA strukturo. To storimo tako, da v vhodno datoteko, ki je formata 2.5.1, najprej vnesemo našo QCA strukturo in nato parametre simulacije. Na sliki 2.10 vidimo primer trojiškega negatorja, kot bi izgledal v našem orodju. Za tako QCA strukturo ustvarimo vhodno datoteko 2.2. Vrstica 1 in 2 določata arhitekturo, celice so naštete med vrsticami 5 in 9, parametri simulacije so med vrsticami 11 in 21, in na koncu je sekvenca vhodnih stanj, med vrsticami 24 in 27.

Listing 2.2: Vhodna datoteka `input.qdStruct` za simuliranje trojiškega negatorja.

```
1  5
2  60.000000 1 1 1
3
4
5  T E 30.000000 30.000000 15.000000 U A
6  T D 90.000000 30.000000 15.000000 H N
7  T I 150.000000 30.000000 15.000000 H N
8  T I 210.000000 90.000000 15.000000 S N
9  T O 270.000000 90.000000 15.000000 S N
10
11 1
12
13 50
14
15 1e-05
16
17 100
18
19 INF
```



Slika 2.10: Trojiški negator z eno celico z vgrajenim stanjem, eno vhodno, dvema delovnima in eno izhodno celico.

```
20
21 4
22
23
24 A
25 B
26 C
27 A
```

Simulator vrne pričakovano datoteko, kjer so v eni vrstici podatki enega stanja sistema. Podatki so zaporedje števil in znakov, ločeni s tabulatorji.

Poglavje 3

Orodje QCA

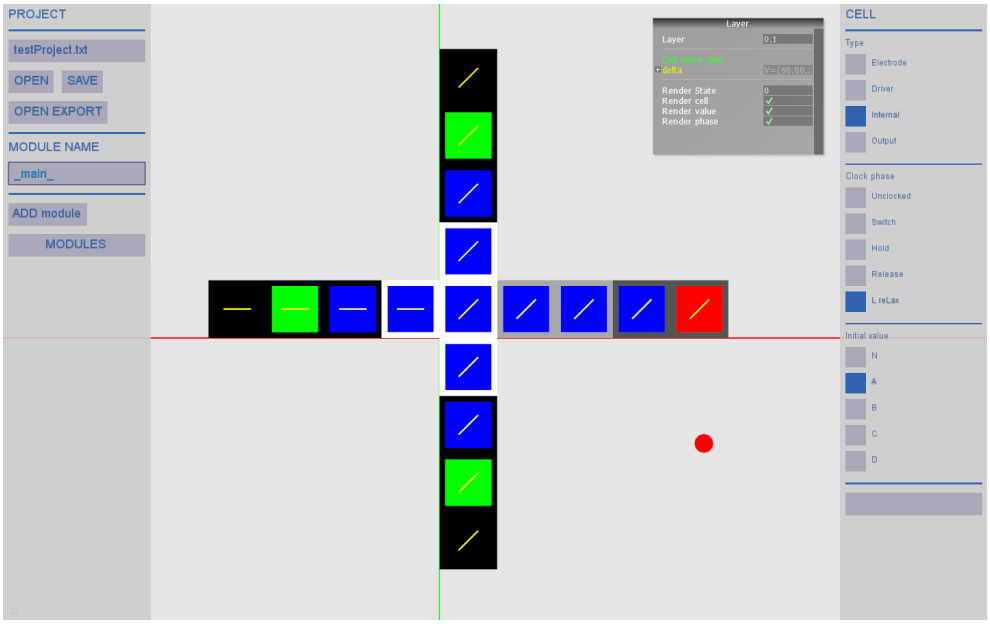
V tem poglavju bomo predstavili cilj diplomske naloge, ki je razvoj programskega orodja. Orodje naj bi omogočalo raziskovanje QCA struktur, ki temeljijo na tehnologiji, opisani v prejšnjih dveh poglavjih.

3.1 Pregled funkcionalnosti

Orodje je sestavljeno na podlagi uporabe qdCAD simulatorja, opisanega v poglavju 2.5. Ta za svoj vhod sprejme datoteko, ki vsebuje QCA strukturo in parametre simulacije. Rezultat simulatorja je datoteka, v kateri so opisana stanja celic skozi celotno simulacijo. Orodje sestoji iz dveh delov. Prvi omogoča posreden in pregleden način urejanja vhodne simulacijske datoteke, to so sestavljanje QCA strukture in spreminjanje parametrov simulacije. Drugi del orodja pa omogoča pregledno predstavitev rezultatov simulacije. Program omogoča tudi shranjevanja notranjega stanja ter komunikacijo s simulatorjem. Ti glavni deli orodja so predstavljeni v naslednjih poglavjih.

3.2 Urejanje strukture

Ta del programa nam omogoča urejanje poljubne QCA strukture. Celice so predstavljene v 3D svetu, kot kocke različnih barv in oblik, kot vidimo na

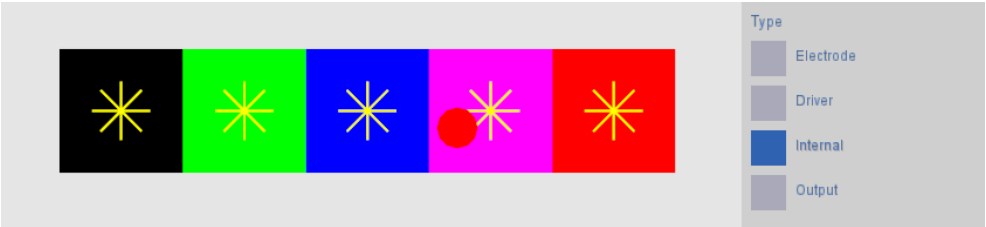


Slika 3.1: Primer urejene QCA strukture (trojiška majoritetna vrata) v našem orodju.

sliki 3.1. Te lastnosti kock odražajo parametre QCA celice:

- Pozicija. Določa pozicijo celice v QCA strukturi. Je sestavljena iz 3 float koordinat. X in Y določata pozicijo na ravnini X Y, medtem ko Z koordinata predstavlja plast (angl. *layer*), na kateri se nahaja celica. Spreminjamo jo z držanjem Shift tipke in miško. V tem primeru so pozicije diskretizirane na večkratnik širine ali, v primeru plasti, na višino celice. Označeno celico lahko premikamo tudi za poljubno število. Pri tem smerne tipke premaknejo označeno celico za vrednosti, ki so shranjene v spremenljivki *Delta*, ki jo vidimo v uporabniškem vmesniku.

- Tip (angl. *type*). Odraža tip celice, opisan v poglavju 2.4. Predstavljen je z barvo dna kocke. Črna - celica z vgrajenim stanjem, zelena - vhodna celica, modra - delovna celica, rdeča - izhodna celica. Na sliki 3.2 so celice različnih tipov.



Slika 3.2: Prikaz različnih tipov celic in označene delovne celice. Od leve: celica z vgrajenim stanjem, vhodna celica, delovna celica, označena delovna celica, izhodna celica. Na desni je del uporabniškega vmesnika, ki omogoča spreminjanje tipa označene celice.

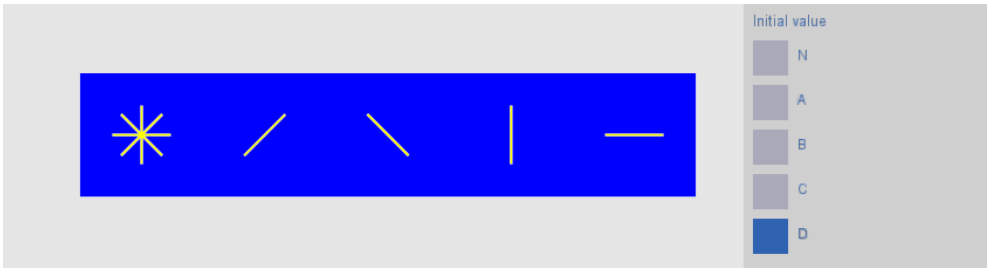
- Začetna vrednost (angl. *initial value*). Predstavlja začetno polarizacijo celice, opisano v poglavju 2.3.3. Možne vrednosti so A, B, C, D, ter N - nevtralno. Grafično je predstavljena kot črte na dnu kocke, pri katerem črta označuje diagonalo, na kateri je močna polarizacija, ali zvezda črt, kar označuje nevtralno polarizacijo, kot je prikazano na sliki 3.3.

- Urina faza (angl. *clock phase*). Opisana v poglavju 2.3.1. Predstavljena je kot različno obarvan rob kocke, prikazano in opisano na sliki 3.4.

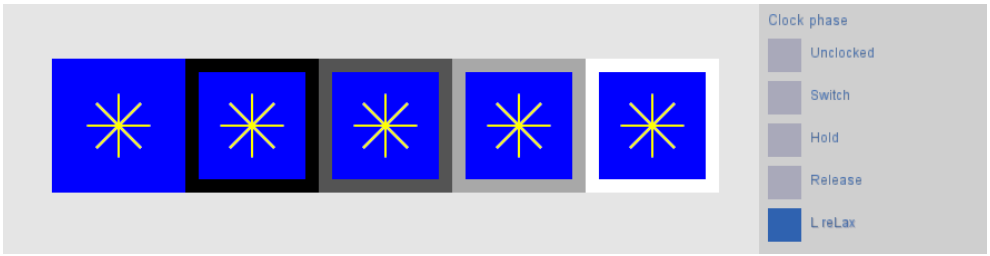
Spreminjanje parametrov celic poteka preko uporabniškega vmesnika na levi strani ter neposrednega klikanja na celice ter držanja ustreznih tipk. V uporabniškem vmesniku so tudi stikala, ki označujejo katere parametre celic želimo izrisati. Tako lahko izberemo na primer samo izris urinih faz celic.

3.2.1 3D izbiranje

Implementirana je metoda 3D izbiranja (angl. *3D picking*). Ta pretvori koordinate miškinega klika v koordinate 3D sveta. Pozicija miške na zaslonu predstavlja premico v 3D prostoru. Pozicijo miške nato izračunamo kot presečišče premice in ravnine, ki je v našem primeru plast, na kateri ležijo celice. Metoda je uporabljena pri označevanju in premikanju celic.



Slika 3.3: Celice z različnimi začetnimi polarizacijami, opisanimi v poglavju 2.3.3, in del uporabniškega vmesnika, ki spreminja začetne vrednosti celic. Od leve so celice s polarizacijami: $NABCD$.



Slika 3.4: Prikaz celic z različnimi fazami, kar se kaže kot različno obarvan rob celice. Gledano iz leve so celice v fazah: brezfazna celica, preklopa, zadrževanja, sproščanja, sproščenosti.

Ko označimo celico, se njeni parametri pokažejo v uporabniškem vmesniku na desni strani. Ta ima gumbe, ki omogočajo spreminjanje parametrov označene celice. Spreminjanje parametrov celice je omogočeno tudi preko črk tipkovnice:

- RTZU - tip celice. R - celica z vgrajenim stanjem, T - vhodna, Z - delovna, U - izhodna.
- FGHJK - urina faza celice (F - brezfazno stanje, G - faza preklopa, H - faza zadrževanja, j - faza sproščanja, K - faza sproščenosti
- CVBNM - začetna vrednost celice. C - nevtralno stanje, V - A, B - B, N - C, M - D.

3.2.2 Moduli

Logična organizacija strukture je zasnovana na ideji modulov. Modul predstavlja eno logično strukturo, npr. AND vrata. Sestavljen je iz skupka množice QCA celic in množice modulov (ali *podmodulov*), ter njihovih pozicij. Tako dobimo drevo modulov. Pri gradnji drevesa moramo paziti na možnost ciklov, katerim se izognemo z rekurzivnim preverjanjem, ali podmodul vsebuje modul, kateremu ga želimo dodati. Premikanje modulov poteka isto kot premikanje celice, le da tukaj premaknemo celoten modul. Vsak modul je v svojem koordinatnem sistemu, kar pomeni da se moramo v primeru drevesne strukture premikati skozi njih. Tako se ob izrisu in izvozu celicam podmodula prišteva pozicija podmodula.

Nov modul se ustvari z tipko CTRL+N. Izbira in dodajanje modula je možno v uporabniškem vmesniku na levi strani. Na sliki 3.1 vidimo primer enega samostojnega modula. Na slikah 3.9 in 3.10 vidimo dva samostojna modula, ki jih lahko grupiramo v nov modul, kot vidimo na sliki 3.12.

3.3 Simuliranje

Simuliranje QCA strukture se izvede z zunanjim, poljubnim simulatorjem. V tej verziji orodja je uporabljen simulator qdCAD, opisan v poglavju 2.5. Ta se izvaja v ločeni niti, kajti simuliranje pogosto traja več kot 10s. Komunikacija s simulatorjem poteka preko vhodne in izhodne datoteke, opisane v poglavju 2.5. Urejanje parametrov simulacije je narejeno v uporabniškem vmesniku, ki vsebuje polja za vsak parameter, ter uredljiv seznam vhodnih stanj.

Pri ustvarjanju vhodne datoteke izvozimo samo trenutno izbrani modul in rekurzivno njegove podmodule. Pozicijam celic podmodula prištejemo pozicijo podmodula v glavnem modulu.

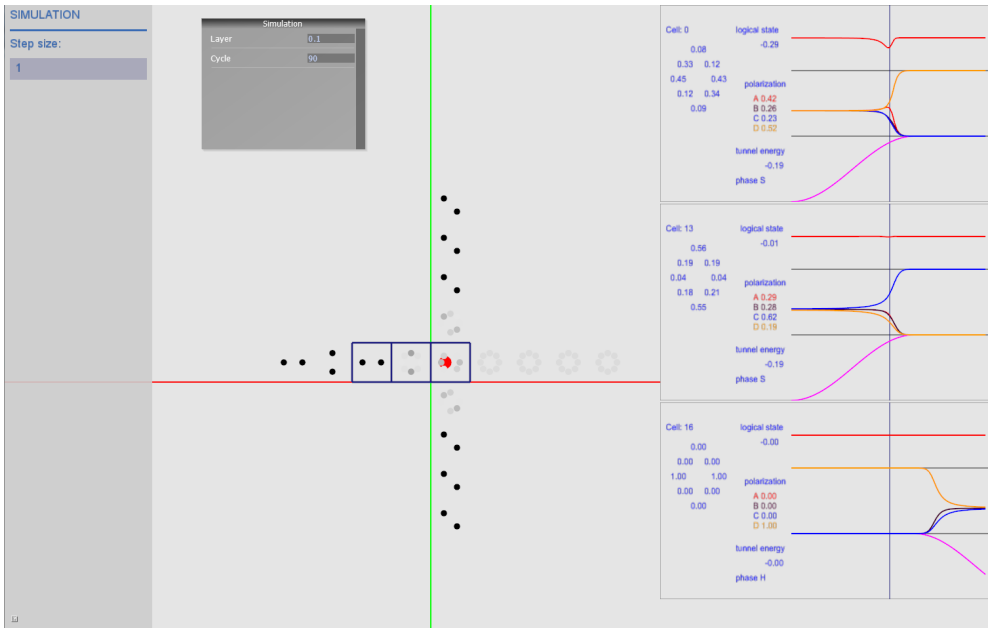
Gumba za izvoz vhodne datoteke in simulacijo sta ločena, kot je gumb za uvoz izhodne datoteke.

3.4 Predstavitev rezultatov

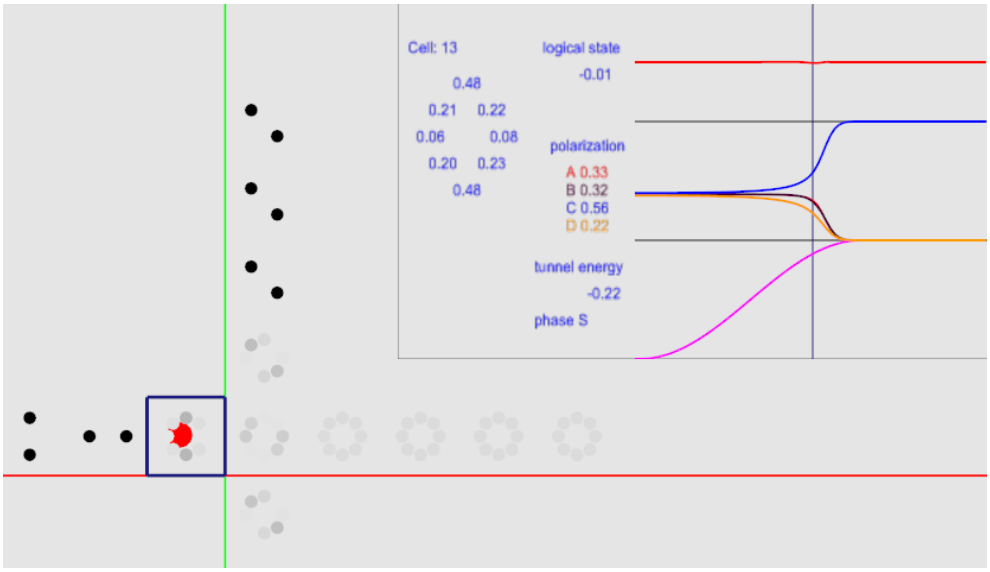
Predstavitev rezultatov je odvisna zgolj od izhodne datoteke simulatorja. Pri uvozu se podatki shranijo v strukturo, opisano v poglavju 4.1.2. Ta, podobno kot izhodna datoteka, ločuje podatke glede na časovni korak simulacije. En cikel simulacije predstavlja neodvisno, statično predstavitev sistema. Grafični pogon tako zmeraj statično izrisuje določen cikel simulacije. Podobno kot pri urejanju strukture, smo se naslonili na predstavitev podatkov preko geometričnih oblik z različnimi lastnostmi.

Grafični pogon NABOJ predstavi kvantne pike, v obliki krogel, katerih barva in prosojnost predstavlja ρ vrednost pik. Črna, neprosojna barva pomeni visok ρ, bela, prosojna barva pa pomeni nizek ρ. Primer trojiških majoritetnih vrat na sliki 3.5.

Grafični pogon omogoča tudi podroben pogled v podatke posamezne pike, slika 3.6. Izbira pik poteka preko 3D izbiranja, kjer se pika, ki je najbližja kliku doda v množico izbranih. Na zaslonu se nato izpišejo parametri celice v tistem časovnem koraku, opisani v poglavju 2.3. Izrišejo se tudi grafi vrednosti parametrov celice v odvisnosti od časovnega koraka. To so logično



Slika 3.5: Prikaz rezultatov simulacije majoritetnih vrat z grafičnim pogonom naboj. Označeni sta dve celici, katerih parametre vidimo izpisane in izrisane v grafih na desni strani.



Slika 3.6: Na levi strani prikaz vseh parametrov izbrane celice v tem urinem ciklu. Na desni graf strani trije grafi: logično stanje - L, polarizacija diagonal - P in tunelirna energija - T.

stanje - L, polarizacija diagonal - P, in tunelirna energija - T.

3.5 Primer uporabe

V tem poglavju bomo prikazali primer uporabe našega orodja za simuliranje QCA struktur. Najprej si zamislimo makro obliko strukture, ki jo želimo preizkusiti. Ker smo navajeni dvojiških logičnih funkcij, bomo kar naivno poizkusili narediti trojiško strukturo, ki preslika dva vhoda X in Y v izhod Z. Pri tem bomo oba vhoda in izhod negirali. Torej bomo realizirali preslikavo $\neg(\neg X \vee \neg Y) \rightarrow Z$. Naše orodje omogoča izdelavo modulov, opisani v poglavju 3.2.2, ki predstavljajo neodvisne enote ali QCA strukture. V našem primeru bomo potrebovali dve enoti, modul *not* in modul *or*. Vsak modul bo imel en vhod in en izhod, celice v njem pa bodo razporejene tako, da bodo realizirale želeno preslikavo svojega vhoda v izhod. V orodju modul ustvarimo s pritiskom CTLR + N. Pri tem se odpre nova prazna površina.

Celice, ki jih bomo ustvarili na tej površini bodo del novega modula. V uporabniškem vmesniku na levi strani ga poimenujmo *not*. Tako imamo modul z imenom "NOT", ki ne vsebuje celic, prikazan na sliki 3.7.

Sedaj modul napolnimo s celicami, ki bodo pravilno realizirale želeno preslikavo, v tem primeru negacijo. Celice ustvarimo tako da kliknemo na želeno mesto na zaslonu med držanjem tipke CTRL. Za operacijo negacije bomo potrebovali 7 celic, ki jih postavimo na zaslon, kot prikazano na sliki 3.8.

Naslednji korak je določitev urinih faz celic, opisane v poglavju 2.3.1. Pri tem moramo paziti, da bo modul pravilno sinhroniziran z ostalimi moduli. To rešimo z dogovorom, da bomo imeli vse vhodne celice v fazi preklopa, torej bodo sprejemale vrednost iz okolice. Iz tega sledi, da morajo biti izhodne celice v fazi zadrževanja. Torej bodo sporočale vrednost vhodnim celicam modula, ki bo v strukturi sledili temu. Faze ostalih delovnih celic nato nastavimo tako, da omogočajo razširjanje vrednosti od vhoda proti izhodu. Pri tem bomo naredili en sprehod čez vse štiri faze v pravilnem vrstnem redu: faza preklopa (vhodna celica), faza sproščenosti, faza sproščanja, faza zadrževanja (izhodna celica). Tako dobimo končan modul, kot vidimo na sliki 3.9.

Po istem postopku nato zgradimo modul *or*, ki ima dva vhoda, en izhod, in eno celico z vgrajenim stanjem. Realiziran je z majoritetnimi vrati, kjer je tretji vhod celica z vgrajenim stanjem, kateri bomo določili začetno vrednost C, da dobimo želeno obnašanje modula. Končan modul *or* je prikazan na sliki 3.10.

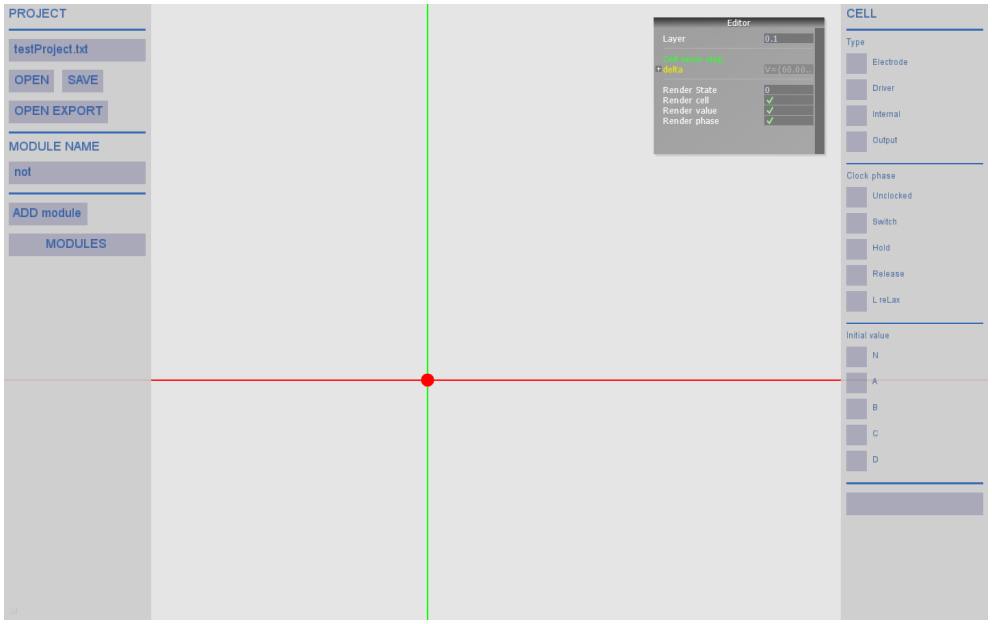
Sedaj v uporabniškem vmesniku izberemo prvi modul, z imenom *_main_*, v katerega bomo dodali 3 module *not* in 1 modul *or* ter jih ustrezno povezali, da bomo ustvarili želeno funkcijo. Modul dodamo tako, da kliknemo, in s tem aktiviramo gumb *ADD module* in nato kliknemo na modul, ki ga želimo dodati. Modul se bo premaknil na pozicijo zadnjega klika miške, ki je označen z rdečo kroglo. Označeni modul lahko premikamo z miško in držanjem gumba SHIFT. Module sedaj ustrezno razporedimo, kot vidimo na sliki 3.11. Pri

tem se moduli med sabo dotikajo samo z vhodnimi in izhodnimi celicami.

V naslednjem koraku poleg vsake od vhodnih celic na levi strani dodamo celico z vgrajenim stanjem. Stanja teh dveh celic bomo med simulacijo spreminjali s sekvenco vhodnih stanj. Začetno vrednost jim določimo enako kot bo prva kombinacija v sekvenci vhodnih stanj. S tem je naša QCA struktura dokončana, kot prikazano na sliki 3.12.

Sedaj ko imamo končano QCA strukturo, moramo pred simulacijo samo še urediti parametre simulacije in sekvenco vhodnih stanj, opisano v poglavju 2.5.1. Uporabniški vmesnik, ki to ureja, odpremo z gumbom *OPEN EXPORT*. V naši strukturi imamo 3 celice z vgrajenim stanjem. Torej bo vsaka kombinacija vhodnih stanj dolga 3 znake. Prvi dve vrednosti se bodo spreminjale, tretja ostane zmeraj na C, saj določa stanje celice, ki je del modula *or*. Na sliki 3.13 vidimo urejene parametre simulacije in sekvenco vhodnih stanj. Ko smo uredili vse potrebne podatke z gumbom *EXPORT* ustvarimo vhodno datoteko simulatorja v katero izvozimo podatke, ki smo jih do sedaj urejali. Pri tem se izvozi samo trenutno odprti modul. Nato z gumbom *SIMULATE* poženemo simulator nad to datoteko. Ko v uporabniškem vmesniku na desni strani dobimo obvestilo *Simulation ended: 0*, se je simulacija končala in lahko uvozimo rezultate simulacije s klikom na gumb *SHOW SIM RESULTS*. Projekt lahko kadar hočemo shranimo s klikom na gumb *SAVE*. Datoteka tega projekta je v poglavju 3.6.

Po uvozu rezultatov simulacije lahko z gumbom F2 preklopimo v drug del našega programa, kjer te podatke pregledamo, vidno na sliki 3.14. Na sliki vidimo prvo stanje sistema. Med vsemi stanji sistema se lahko sprehodimo s smernimi tipkami *levo* in *desno* ali z direktnim nadzorom spremenljivke preko sivega uporabniškega vmesnika *Simulation*. S klikom na površino se označi celica, ki je najbližja kliku. Za označene celice se na desni strani izpišejo vse vrednosti celice ter grafi prehajanj določenih parametrov teh celic. Na sliki 3.15 vidimo dve označeni celici in grafi njunih parametrov. Celici sta trenutno v fazi preklopa. Vidimo lahko kako se preko celic razširja vrednost proti desni. V teh dveh celicah se izvede dejanska negacija vrednosti, ki pa



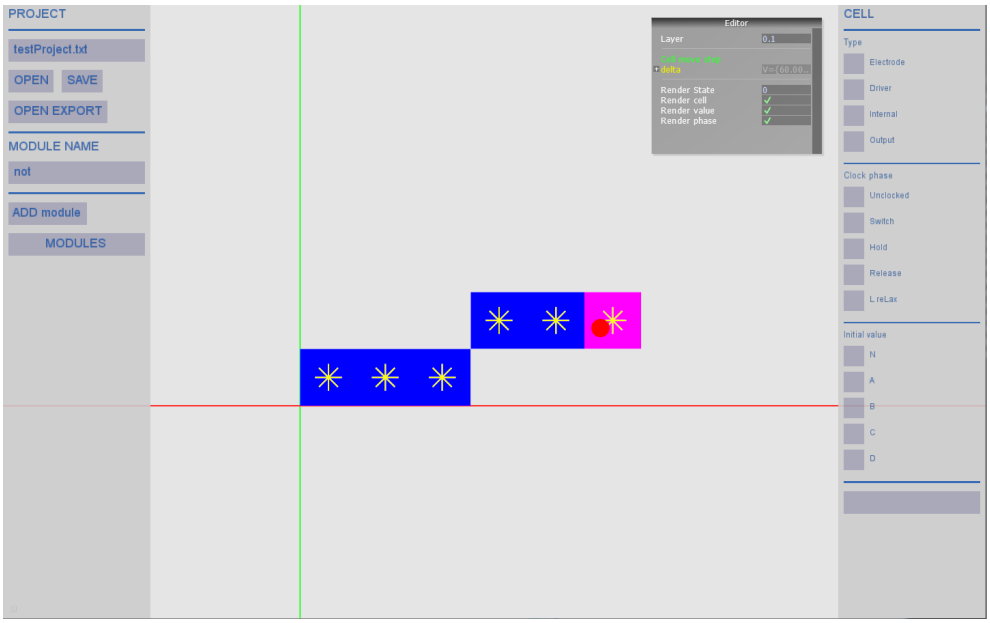
Slika 3.7: Vidno stanje našega programa, ki prikazuje na novo ustvarjen modul z imenom *not*.

sicer niso prave glede na celico z vgrajenim stanjem. Interpretacija rezultatov je sicer bolj problem raziskovalca, ki naše orodje uporablja.

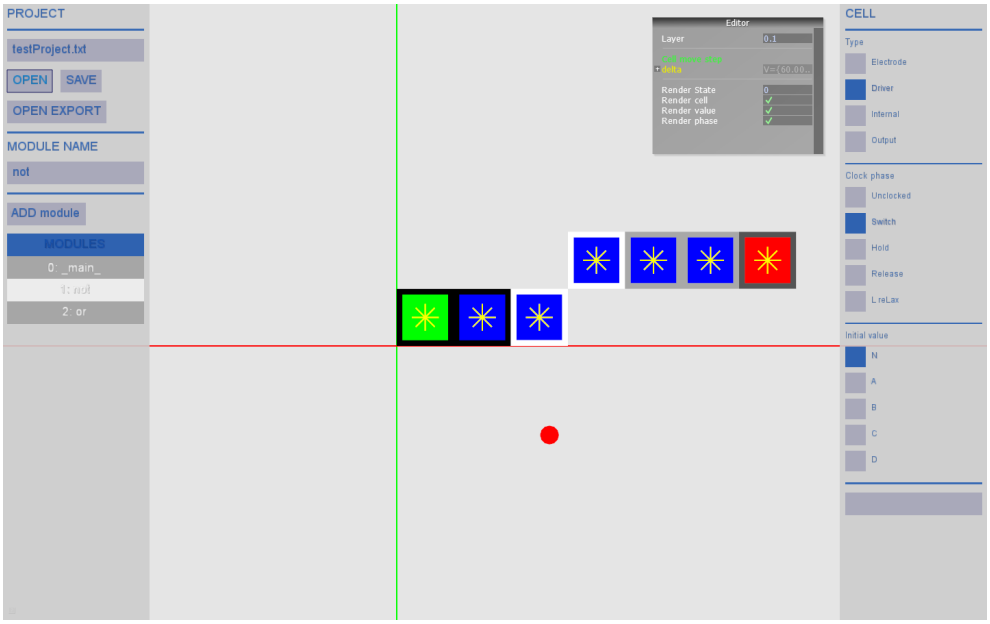
3.6 Shranjevanje stanja

Orodje omogoča shranjevanje in obnavljanje stanja. Pri tem se shranijo podatki QCA strukture, ki jo urejamo, vsi moduli, ter parametri simulacije. Shrani se v datoteko s formatom, prikazana v izpisu 3.1. Datoteka za shranitev stanja za primer, ki smo ga opisali v poglavju 3.5, je prikazan v izpisu 3.2. Razlaga posameznih polj je sledeča:

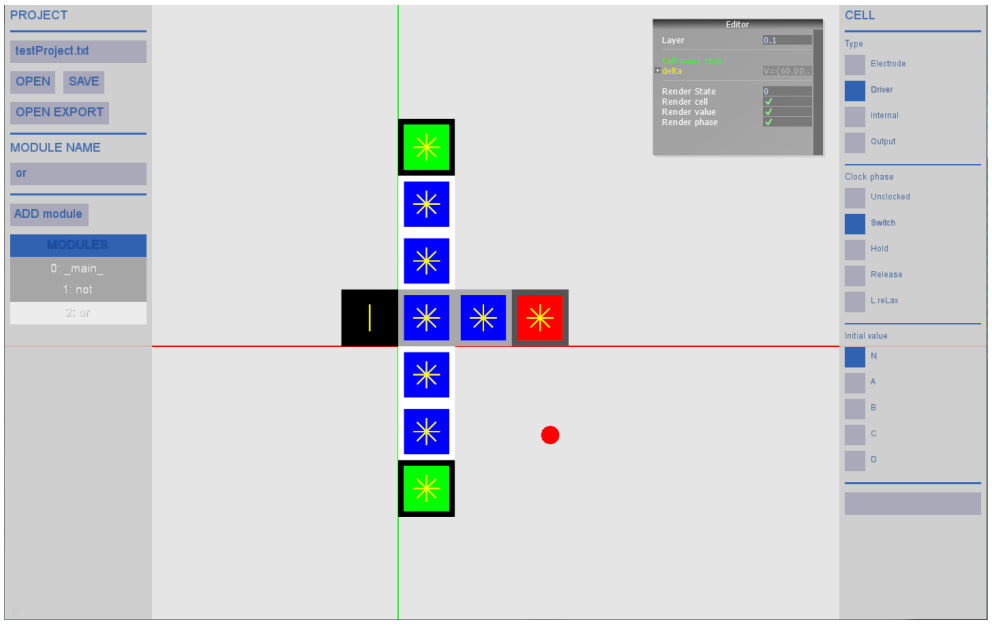
- `cellWidth cellHeight nDots`
  Realni vrednosti širina celice `cellWidth` in višina celice `cellHeight`. `nDots` označuje število pik v celici. Zavzema lahko vrednosti 4, ki predstavlja dvojiško celico, ali 8, ki predstavlja trojiško celico.



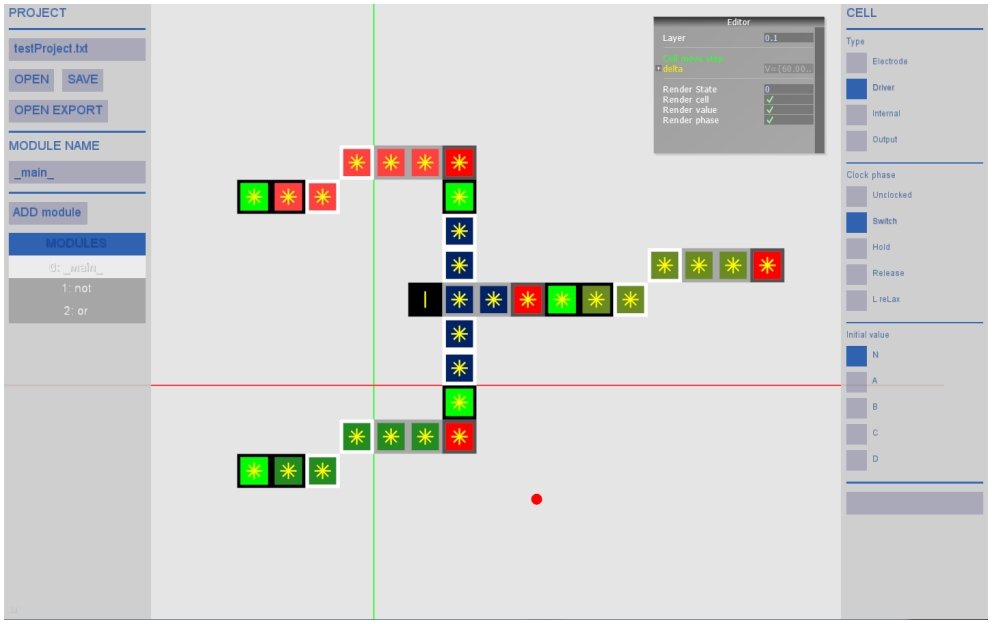
Slika 3.8: Modul *not* z dodanimi sedmimi celicami.



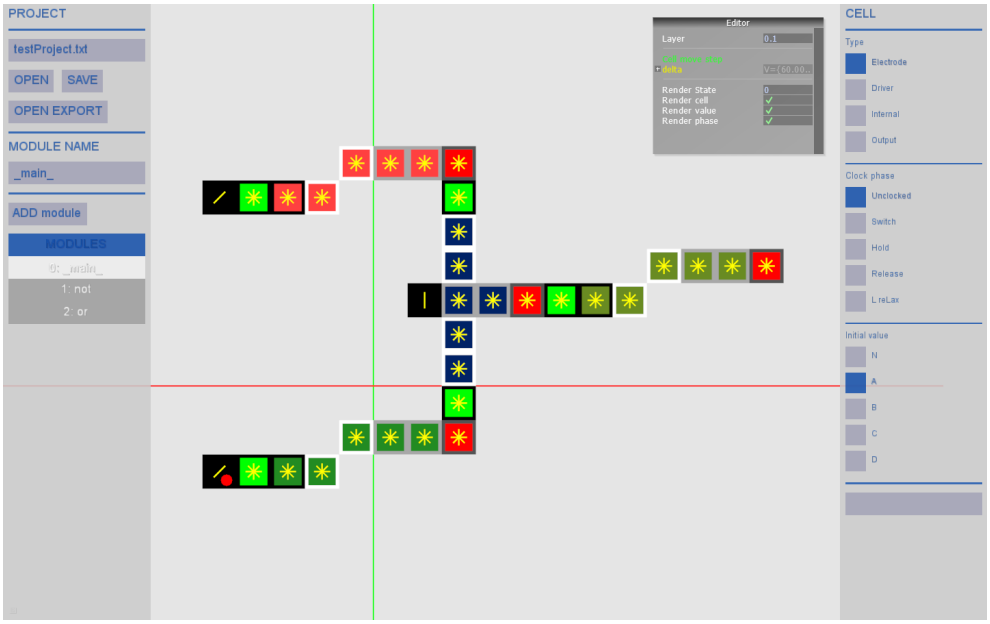
Slika 3.9: Končani modul *not* s sedmimi celicami s pravilno določenimi urinimi fazami.



Slika 3.10: Končani modul *or*, ki sprejme dva vhoda.



Slika 3.11: Modul *_main_*, kateremu smo dodali in ustrezno razporedili tri *not* in en *or* modul.



Slika 3.12: Glavni modul, kateremu smo dodali dve celici z vgrajenim stanjem (najbolj levi črni celici).

- %"`moduleName`"
  Ime modula, katerega celice in podmodule bomo našteli.

- `type mode x y z phase value`
  Vsaka vrstica opisuje eno QCA celico s parametri, opisani v poglavju 2.3.
  `type` določa ali je celica dvojiška - B ali trojiška - T.
  `mode` določa tip celice: celica z vgrajenim stanjem - E, vhodna celica - D, delovna celica - I, izhodna celica O.
  `x y z` so realne vrednosti, ki določajo pozicijo celice.
  `phase` določa začetno urino fazo celice: brezfazno stanje - U, faza preklopa - S, faza zadrževanja - H, faza sproščanja - R, faza sprostitve - L.
  `value` določa začetno vrednost celice: $NABCD$.

- "`moduleName x y z`"

Slika 3.13: Urejeni parametri simulacije in želena sekvenca vhodnih stanj za naš primer uporabe orodja.

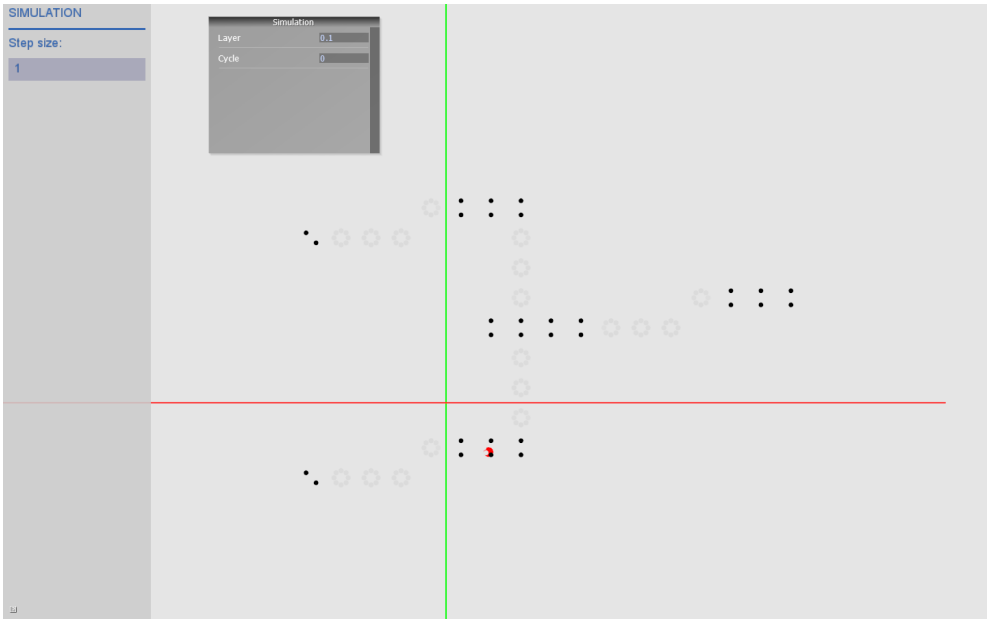
Ime podmodula `moduleName` in koordinate njegove pozicije `x y z`, ki so realne vrednosti.

- `nClocks`
  Celoštevilska vrednost, ki določa število urinih period simulacije.

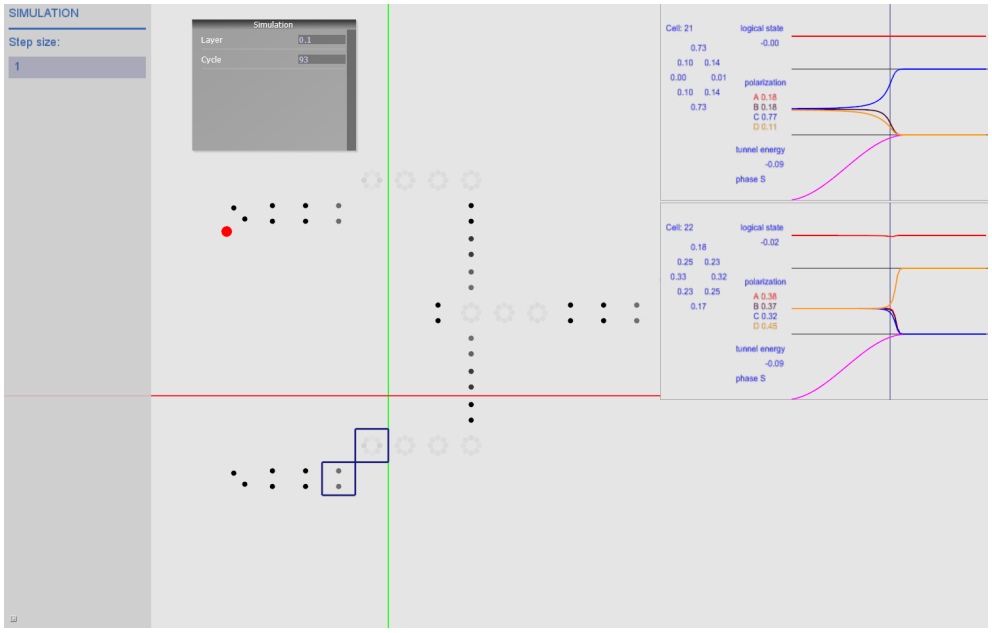
- `ticsPerQuarterTock`
  Celoštevilska vrednosti, ki določa število iteracij ali število računanj novih stanj sistema na eno urino fazo.

- `eps`
  Realna vrednost, ki določa minimalno dovoljeno spremembo energije sistema znotraj ene iteracije simuliranja.

- `nSteps`
  Celoštevilska vrednost, ki določa maksimalno število korakov do dosege minimalne dovoljene spremembe energije sistema.



Slika 3.14: Prvi cikel ali prvo stanje sistema, ki je rezultat simulacije naše strukture. Pri vrhu vidimo tudi uporabniški vmesnik *Simulation* za neposreden nadzor spremenljivke, ki določa kateri cikel simulacije želimo imeti prikazan *Cycle*.



Slika 3.15: Stanje sistema v 93 koraku. Označeni sta dve celici, kjer se izvede dejanska negacija vrednosti, ki potuje skozi strukturo.

- `maxR`
  Realna vrednost, ki določa radij vpliva Coulombove sile .

- `nInputValues`
  Celoštevilska vrednost, ki določa število sekvenc vhodnih stanj.

- *nElectrodes* ∗ `inputValue`
  Sekvenca vhodnih stanj. Določa stanje vsake celice z vgrajenim stanjem vsako urino periodo. Zaloga vrednosti je $ABCD$.

Listing 3.1: Format datoteke za shranjevanja stanja `testProject.txt`

```
1  cellWidth cellHeight nDots
2
3  \%"moduleName"
4
5  type mode x y z phase value
6
7  "moduleName" x y z
8
9  --END--
10
11
12 nClocks(long)
13 ticsPerQuarterTock(long)
14 eps(double)
15 nSteps(long)
16 maxR(double)
17 nInputValues(long)
18
19 s$nElectrodes~*$ inputValue(char(A/B/C/D])
```

Listing 3.2: Datoteka za shranjevanje stanja `testProject.txt` za naš primer, opisan v poglavju 3.5.

```
1  60.000000 30.000000 8
```

```
2
3  %"_main_"
4  T E -300.000000 300.000000 0.000000 U A
5  T E -300.000000 -180.000000 0.000000 U N
6
7  "or" 120.000000 120.000000 0.000000
8
9  "not" -240.000000 300.000000 0.000000
10
11 "not" -240.000000 -180.000000 0.000000
12
13 "not" 300.000000 120.000000 0.000000
14
15 %"not"
16 T D 0.000000 0.000000 0.000000 S N
17 T I 60.000000 0.000000 0.000000 S N
18 T I 120.000000 0.000000 0.000000 L N
19 T I 180.000000 60.000000 0.000000 L N
20 T I 240.000000 60.000000 0.000000 R N
21 T O 360.000000 60.000000 0.000000 H N
22 T I 300.000000 60.000000 0.000000 R N
23
24 %"or"
25 T I 0.000000 0.000000 0.000000 R N
26 T I 0.000000 60.000000 0.000000 L N
27 T I 0.000000 -60.000000 0.000000 L N
28 T I 0.000000 -120.000000 0.000000 L N
29 T I 0.000000 120.000000 0.000000 L N
30 T E -60.000000 0.000000 0.000000 U C
31 T I 60.000000 0.000000 0.000000 R N
32 T O 120.000000 0.000000 0.000000 H N
33 T D 0.000000 180.000000 0.000000 S N
34 T D 0.000000 -180.000000 0.000000 S N
```

```
35
36
37 –END–
38
39 4
40
41 50
42
43 1e-05
44
45 100
46
47 INF
48
49 3
50
51
52 A A C
53 C C C
54 C A C
```

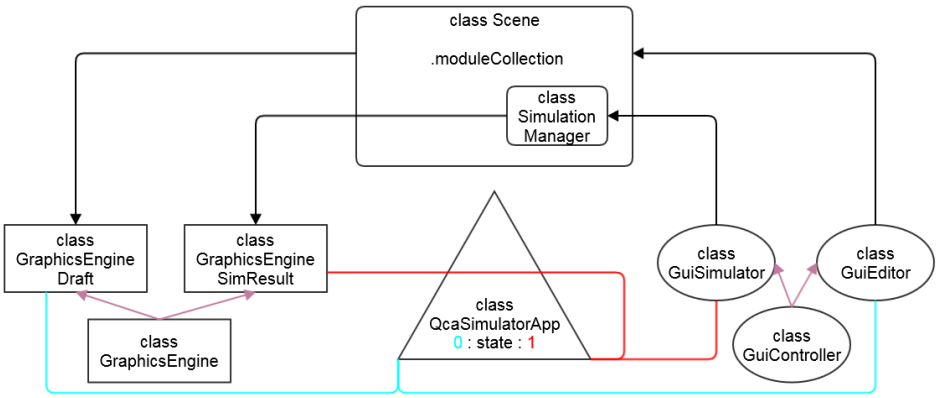
Poglavje 4

Arhitektura programa

Ogrodje programa je del knjižnice Cinder, opisan v poglavju 4.2. Konkretno je ogrodje razred cinder/app/AppNative. Ta poskrbi za ozadje programa, kot so ustvarjanje okolja, lovljenje dogodkov vhodno izhodnih naprav, ter izvajanje glavne zanke (angl. *main loop*) programa. Glavna zanka je osnovnega tipa *setup* → *update* → *draw*. Ta se v knjižnici kaže kot razred AppNative, katerega podedujemo in mu prepišemo (angl. *override*) potrebne metode. Metode se kličejo v skladu z arhitekturo glavne zanke:

- `setup();` Zgradimo objekte scene in ostale objekte naše arhitekture programa.
- `processInput();` Je sestavljena iz metod `mouseDown()` in `keyDown()`. Obdelamo uporabnikov vnos.
- `update();` Posodobimo stanje programa.
- `draw();` Izrišemo sceno in uporabniški vmesnik.

Jedra metod služijo kot oddajnik dogodkov v našo arhitekturo, ki poskrbi za funkcionalnost našega orodja. Od tu naprej je razvita naša lastna arhitektura, vidna na sliki 4.1. Temelji na izmenljivosti posameznih komponent, kot so grafični pogoni, uporabniški vmesniki ter več manjših komponent.



Slika 4.1: Arhitektura našega dela programa. Črne puščice kažejo pretok podatkov, vijolične smer dedovanja, modra in rdeča so stanje programa (0: urejanje; 1: simuliranje)

.

Vodilni konstrukt je scena, opisana v poglavju 4.1.1. Ta je od drugih delov arhitekture neodvisna v smislu, da jih ne pozna, jim ne spreminja delovanja. Služi kot odlagališče lastnosti strukture QCA, rezultatov simulacije in spremenljivk, ki določajo stanja drugih komponent arhitekture.

Drugi konstrukt je uporabniški vmesnik, opisan v poglavju 4.1.3. Je neodvisen od drugih komponent. Obdeluje uporabnikov vnos in ustrezno spreminja sceno ter, preko nje, način prikaza grafičnih pogonov.

Grafični pogoni grafično predstavijo stanje strukture QCA, ki se nahaja v sceni. Med sabo so neodvisni in izmenljivi. Svoje delovanje spreminjajo glede na določene vrednosti v sceni, ki jih spreminja uporabniški vmesnik.

V enem trenutku je lahko aktiven samo en uporabniški vmesnik ter en grafični pogon. Izmenjava teh se dogaja v glavnem razredu `class QcaSimulatorApp`, ki je otrok razreda `class AppNative`.

4.1 Opis komponent

4.1.1 Scena

Scena (`class Scene`) je jedro naše arhitekture. Hrani vse podatke, ki so del simuliranja strukture. Ima dva glavna dela. Prvi je seznam modulov (`vector Scene.moduleCollection`), kar predstavlja celotno strukturo QCA, drugi je urejevalnik simulacije (`class SimulationManager`), ki uvozi in shrani rezultate simulacije. Scena vsebuje metode, ki označujejo in premikajo module (del metode 3D izbiranja, opisan v poglavju 3.2.1), metode za izvoz in uvoz projekta, opisane v poglavju 3.6 ter metode za izvoz simulacijske datoteke, opisan v poglavju 2.5.

Modul (`class Module`) je konstrukt v obliki drevesa, ki hrani sceno celic ter seznam podmodulov in njihovih pozicij. Vsebuje metode za dodajanje celic in modulov, ter izvoz stanja.

Scena celic (`class CellScene`) hrani pozicijo in vse parametre celic, opisane v poglavju 2.3. Njene metode obravnavajo dodajanje in brisanje celic. Lastnosti celice, torej ali so celice dvojiške ali trojiške in njihove dimenzije so shranjene v spremenljivki scene `char Scene.cellType`.

4.1.2 Scena rezultatov simulacije

Scena rezultatov simulacije `class SimulationManager` se v glavni sceni nahaja v `SimulationManager Scene.simRes`. Je razred, ki hrani rezultate izvedene simulacije strukture QCA. Podatke sprejme v izhodni datoteki simulatorja 2.5. Podatke shrani v dva večja seznama:

Prvi, `vector SimulationManager.m_cellDotsPos`, vsebuje x, y, z koordinate vsake pike vsake celice.

Drugi, `vector SimulationManager.m_cycles`, pa je seznam korakov simulacije. En korak opisuje celotno stanje sistema (strukture QCA) v določenem trenutku simulacije.

Razred cikel (`class ClockCycle`) hrani pet seznamov podatkov, ki opi-

šejo stanje sistema v nekem časovnem koraku:

- `m_rho`: (list(float)); Gostota naboja - ρ v vsaki piki vsake celice, opisan v poglavju 2.3.2.
- `m_cellPhase`: (char); Trenutna faza celice, opisana v poglavju 2.4;
- `m_cellTunnelEnergy`: (float); Tunelirna energija celice, opisana v poglavju 2.3.1;
- `m_P`: (list(float)); Gostota naboja na vsaki diagonali celice, opisana v poglavju 2.3.3;
- `m_L`: (float); Logično stanje celice, opisano v poglavju 2.3.4;

4.1.3 Uporabniški vmesnik

Drugi večji del arhitekture je uporabniški vmesnik. Ta obravnava uporabnikov vnos in izvrši ustrezno spremembo v sceni, kot so spreminjanje parametrov celic, njihove pozicije, lokacije modulov, ... Sestavljen je iz več razredov, ki imajo skupnega starša (`class GuiController`). Ločen je zaradi zasnove orodja, ki predvideva večstopenjsko delovanje. Uporabniški vmesnik je narejen s pomočjo knjižnice ciUI, opisano v poglavju 4.3. Del vmesnika je narejen s pomočjo `class Cinder::Params`, ki omogoča neposreden nadzor nad spremenljivkami.

Prvi uporabniški vmesnik je (`class GuiEditor`), ki dela nad urejanjem strukture in vsebuje naslednje funkcionalnosti:

- Ureja podatke strukture QCA. Deloma spreminja strukturo s sporočanjem dogodkov sceni, ki nato poskrbi za ustrezno spremembo, deloma jo spreminja neposredno. Sceni sporoča pozicije miške in njihov namen (premiki, označevanje modulov/celic). Sprememba parametra celice poteka neposredno, saj je sestavljen le iz spremembe znaka, ki označuje lastnost.

- Spreminja parameter grafičnega pogona, ki izrisuje sceno: `int Scene.RenderState`. Ta določa način prikaza celic. Nadzira se s pomočjo `class Cinder::Params`.

- Ureja izvoz in obnovo stanja projekta.

- Sproži izvoz in uvoz simulacijskih datotek, ter izvajanje simuliranja. Pri tem ustvari novo nit, v kateri požene simulator 2.5 kot zunanji program z ukazno vrstico *"qdCAD_sim.exe tmpExport.qdStruct tmpExport.qdSim"*.

Drugi del uporabniškega vmesnika nadzira predstavljanje rezultatov simulacije. Vsebuje elemente za izbiro želenega koraka simulacije in velikost spremembe koraka pri uporabi smernih tipk za premikanje po korakih simulacije.

Prav tako poskrbi za izris podrobnih parametrov celice, kot smo opisali v poglavju predstavitev rezultatov 3.4. To storimo tako, da določimo ustrezno vidno odprtino preko klicev `glViewport()` in `glScissor()`. Nato nastavimo ustrezno ortografsko projekcijo in izrišemo podatke. Parametre, ki so v tekstni obliki izrišemo z klicem `Cinder::gl::drawString()`.

4.1.4 Grafični pogoni

Tretji večji del arhitekture so grafični pogoni. Ti izrisujejo stanje scene na zaslon, s čimer naj bi nudili pregleden vpogled v stanje strukture. Odvisni so samo od scene, v kateri ležijo podatki ter vrednosti, ki določajo način izrisa. Ogrodje grafičnega pogona je določeno v razredu `class GraphicsEngine`. Ta vsebuje dve osnovni metodi, ki jih dedujoči razredi prepišejo: `setup()`, ki inicializira grafični pogon, in `draw(Scene)`, ki ob klicu na zaslon izriše željeno predstavitev scene. Implementirana sta dva grafična pogona, vsak za eno stopnjo programa.

Grafični pogon osnutek (`class GraphicsEngineDraft`) prikazuje stanje strukture QCA, ki jo urejamo. Pri tem uporablja geometrične oblike različnih

lastnosti, kot opisano v poglavju 3.2. Pri inicializaciji se iz datoteke `colors.txt` naložijo barve. V datoteki so zapisane vsaka v svoji vrsti, iz katerih se preberejo prve tri številke med 0 in 255, ki določajo RGB (*Red Green Blue*) vrednost barve. Prvih šest barv je rezerviranih za določene lastnosti celic. Po vrsti določajo barve: označene celice/modula, delovnih celic, celic z vgrajenim stanjem, vhodnih celic, izhodnih, barva diagonal, ki označujejo polarizacijo diagonal. Ostale barve se uporabijo za barve podmodulov.

Prej omenjena spremenljivka `Scene::RenderState` spreminja način izrisa celic. Vrednost spremeni vidnost celic glede na trenutno opazovano plast celic:

- 0 : izrišejo se vse celice;
- 1 : določi prosojnost celice glede na oddaljenost od trenutno izbrane plasti;
- 2 : izrišejo se samo izbrana plast celic in celice, ki so pod trenutno plastjo. Te se izrišejo z progresivno nižjimi prosojnostmi, bolj kot so oddaljeni.
- 3 : izriše samo izbrano plast celic.

Pogon omogoča tudi izrisovanje samo določenih lastnosti celic. Stikala so v uporabniškem vmesniku. Ob izrisu se pozicijam celic podmodula prišteva odmik podmodula glede na njegovega starša.

Drugi pogon se imenuje grafični pogon nabojev (`class GraphicsEngineSimCharge`). Ta je del drugega dela orodja, kjer predstavljamo rezultate simulacije. Podatke dobi iz scene rezultatov simulacije, opisane v poglavju 4.1.2. Kvantne pike izriše kot krogle, katerih barva določa gostoto naboja ρ v piki. Izriše tudi kvadrate okoli celic, ki so izbrane za izpis vrednosti in izris grafov, ki je del uporabniškega vmesnika, opisan v poglavju 3.4.

4.2 Cinder

Cinder je recenzirana, odprto kodna C++ knjižnica za kreativno kodiranje. [7]. Osredotoča se na kodo in algoritme, ki so potrebni za programiranje grafike, zvoka, omrežnih povezav, procesiranja slik in računalniške geometrije.

Vzpostavitev projekta v Visual Studiu je omogočil TinderBox, ki je orodje za integracijo Cinderja z Visual Studiom.

Vodilni razred naše aplikacije je `class QcaSimulatorApp`, ki je otrok razreda `class Cinder::app::AppNative`. Slednji ima implementirano arhitekturo, ki smo opisali na začetku tega poglavja. Skrbi za zadnji del aplikacije (angl. *backend*). Pri inicializaciji ustvari novo okno aplikacije, ki vsebuje risalno površino. V njo rišemo želene grafične elemente, kot so uporabniški vmesnik in QCA strukturo. To lahko storimo neposredno z uporabo grafične knjižnice *C++ OpenGL*, Cinder pa nam nudi tudi svojo nadgradnjo grafične knjižnice, zapisano v datoteki `cinder/gl/gl.h`. Ta preslika strukture in ukaze, zapisane v obsegu Cindra v strukture in ukaze okolja *OpenGL*. Naprimer kamera `class Cinder::Camera`, ki ima projekcijsko in perspektivno matriko shranjeno v objektih razreda `class Matrix44f`. Jedro `gl::setMatrices(Camera)` ustrezno nastavi projekcijsko matriko in matriko kamere v okolju *OpenGL*.

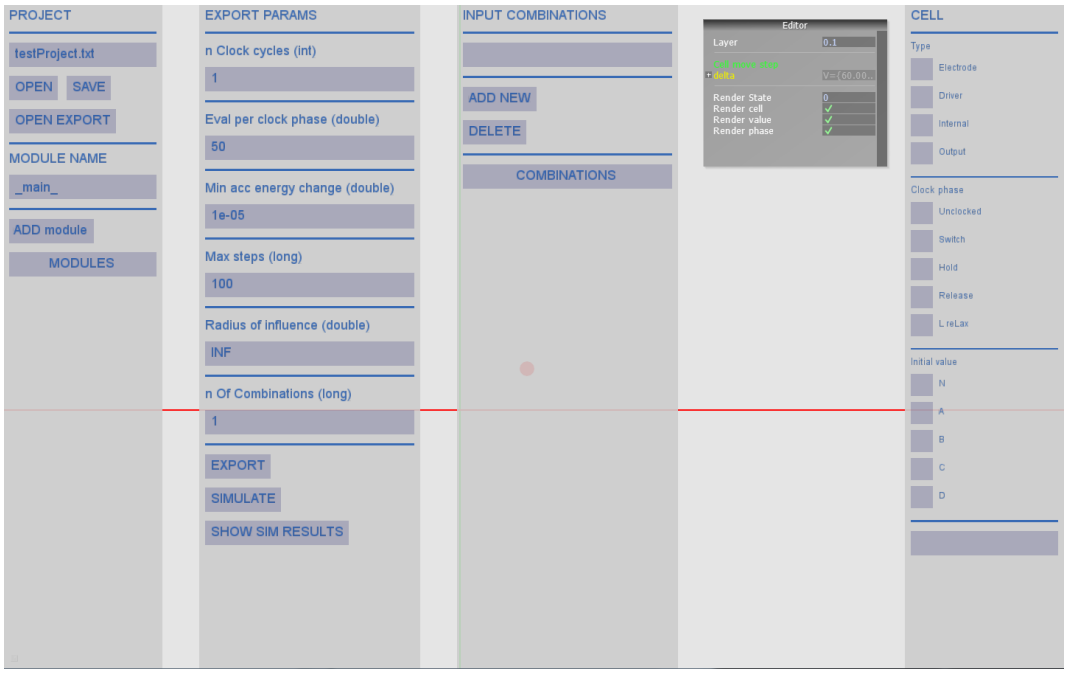
Razred `Cinder::app::AppNative` skrbi tudi za lovljenje dogodkov vhodnih enot, miške in tipkovnice. Te nam sporoči preko metod `AppNative.keyDown(KeyEvent)` in `AppNative.MouseDown(MouseEvent)`. Knjižnica Cinder se nam zdi najbolj uporabna zaradi dobro zastavljenih osnovnih podatkovnih struktur, kot sta vektor `class Vec3f` in matrika `class Matrix44f` različnih velikosti. Implementirano je veliko algoritmov in postopkov za obdelovanje matrik in vektorjev, kot so vse transformacije, transponiranje, množenje matrik, matrik in vektorjev, računanje determinant in tako dalje. Ti postopki so nam pogosto prišli prav pri hranjenju in urejanju podatkov naše scene, ter nadziranju kamere.

Za del uporabniškega vmesnika smo uporabili razred `Cinder::params`, ki omogoča neposreden nadzor nad spremenljivkami, kot so `float` in `bool`

ter tudi kompleksnimi strukturami, implementirani v Cinder, kot so vektor `class Vec3f`. Viden je na sliki 4.2 zgoraj proti desni, z imenom *Editor*.

4.3 ciUI

CiUI [11], avtor Reza Ali, je samostojno razvita knjižnica za implementacijo uporabniškega vmesnika v okolju Cinder. Glavni razred knjižnice je `class ciUICanvas`, imenujmo ga platno, ki predstavlja odložišče za elemente uporabniškega vmesnika. Izriše se kot obarvana podlaga, na kateri se izrisujejo elementi uporabniškega vmesnika, ki mu jih podamo. Te otroke dodamo klicem metode `ciUICanvas.addWidget()`. Platno vsem svojim otrokom nadzira pozicije in barvno temo. Obravnava tudi uporabnikov vnos in odzivne funkcije na določene dogodke. Te ustvarimo sami, platnu ga podamo s klicem metode `ciUICanvas.registerUIEvents(object, void* callback(ciUIEvent * event))`. Pri tem je `void* callback(ciUIEvent * event)` odzivna funkcija poljubnega objekta `object`. Parameter `ciUIEvent * event` vsebuje ime prožilca in tip dogodka. Elementi uporabniškega vmesnika so vsi otroci razreda `class ciUIWidget`. Na sliki 4.2 vidimo uporabniški vmesnik urejevalnika QCA strukture in parametrov izvoza podatkov.



Slika 4.2: Uporabniški vmesnik urejevalnika QCA strukture in parametrov simulacije. Vidimo štiri navpična platna `class ciUICanvas`, na katerih so grafični elementi, kot so gumbi `class ciUILabelButton` in vnosna polja `class ciUITextInput`. Zgoraj desno vidimo tudi uporabniški vmesnik *Editor*, narejen z uporabo `Cinder::params`, ki nadzira parametre izrisa QCA strukture in premikanja celic.

Poglavje 5

Zaključek

Za diplomsko delo smo razvili program, ki omogoča raziskovanje trojiških QCA struktur. Za razumevanje problema smo najprej potrebovali znanje o tehnologiji kvantnih celičnih avtomatov (QCA). Pri tem smo se primarno naslonili na magistrsko nalogo Pečarja [8], ki dobro opiše osnovne gradnike QCA strukture, to so QCA celice. Znanje nam je omogočilo razumevanje simulatorja QCA struktur in komunikacije z njim. Ta za uporablja podatke v surovi obliki. Naše orodje te surove podatke prikazuje na bolj pregleden način preko grafičnih elementov. Ugotovili smo, da simuliranje poteka v treh korakih, za katere smo izdelali posebne dele programa. Ti koraki so urejanje QCA strukture, nato simuliranje in na koncu predstavitev rezultatov simulacije.

Program smo razvili v jeziku C++ v razvojnem okolju Visual Studio. Ogrodje programa je sestavljeno iz knjižnice Cinder, ki ponuja temelj grafične aplikacije. Ta temelj smo nadgradili z lastno arhitekturo programa, ki vsebuje prej opisano potrebno funkcionalnost.

V pričujočem delu smo na koncu predstavili vse pridobljeno znanje o simuliranju QCA struktur in opisali narejeno orodje, njegovo uporabo, arhitekturo in funkcionalnost.

Z našo arhitekturo, ki temelji na izmenljivosti posameznih komponent smo pripravili zadovoljivo podlago za vnaprejšnji razvoj orodja. Nadgradnja

bi temeljila na razvoju grafičnih pogonov, ki bi surove podatke prikazala na druge, različne načine, ter tako nudila drugačen pogled na postopke simuliranja. Potencial vidimo tudi v bolj kompleksnih urejevalnih metodah, ki bi prestavile nivo urejanja na višji nivo. Podlaga v našem orodju je za to tudi primerna, to smo poskusili zagotoviti z modularno zasnovo QCA struktur.

Literatura

[1] C Cohen-Tannoudji, B Diu, and F Laloë. *Quantum Mechanics*, volume 1 - 2. John Wiley & Sons, Paris, 2006.

[2] T Cole and JC Lusth. Quantum-dot cellular automata. *Progress in Quantum Electronics*, 25(4):165–189, 2001.

[3] B Hayes. Computing science: Third base. *American scientist*, 89(6):490–494, 2001.

[4] I Lebar Bajec, N Zimic, and M Mraz. Towards the bottom-up concept: Extended quantum-dot cellular automata. *Microelectronic engineering*, 83(4):1826–1829, 2006.

[5] C S Lent and P D Tougaw. A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85(4):541–557, 1997.

[6] C S Lent, P D Tougaw, W Porod, and G H Bernstein. Quantum cellular automata. *Nanotechnology*, 4(1):49, 1993.

[7] Cinder library. `http://libcinder.org/`.

[8] P Pecar. Uporaba adiabatnega pristopa pri realizaciji trojiškega procesiranja na osnovi kvantnih celicnih avtomatov. Master's thesis, Univerza v Ljubljani, Fakulteta za racunalništvo in informatiko, 2007.

[9] P Pevčar, A Ramšak, N Zimic, M Mraz, and I Lebar Bajec. Adiabatic pipelining: a key to ternary computing with quantum dots. *Nanotechnology*, 19(49):495401, 2008.

[10] W Porod. Quantum-dot devices and quantum-dot cellular automata. *International Journal of Bifurcation and Chaos*, 7(10):2199–2218, 1997.

[11] A Reza. Ciui - user interface block for cinder. `http://www.syedrezaali.com/cinderui/`.

[12] GL Snider, AO Orlov, I Amlani, X Zuo, GH Bernstein, CS Lent, JL Merz, and W Porod. Quantum-dot cellular automata: Review and recent experiments. *Journal of Applied Physics*, 85(8):4283–4285, 1999.

[13] Stephen Wolfram. *A new kind of science*, volume 5. Wolfram media Champaign, 2002.